Data & Knowledge Engineering 70 (2011) 617–641

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Optimizing in-network aggregate queries in wireless sensor networks for energy saving

Chih-Chieh Hung, Wen-Chih Peng *

Department of Computer Science, National Chiao Tung University, Taiwan

ARTICLE INFO

Article history:
Received 10 February 2011
Received in revised form 3 March 2011
Accepted 3 March 2011
Available online 17 March 2011

Keywords:
In-network aggregate query
Sensor data management
Wireless sensor networks

ABSTRACT

This study proposes a method of in-network aggregate query processing to reduce the number of messages incurred in a wireless sensor network. When aggregate queries are issued to the resource-constrained wireless sensor network, it is important to efficiently perform these queries. Given a set of multiple aggregate queries, the proposed approach shares intermediate results among queries to reduce the number of messages. When the sink receives multiple queries, it should be propagated these queries to a wireless sensor network via existing routing protocols. The sink could obtain the corresponding topology of queries and views each query as a query tree. With a set of query trees collected at the sink, it is necessary to determine a set of backbones that share intermediate results with other query trees (called non-backbones). First, it is necessary to formulate the objective cost function for backbones and non-backbones. Using this objective cost function, it is possible to derive a reduction graph that reveals possible cases of sharing intermediate results among query trees. Using the reduction graph, this study first proposes a heuristic algorithm BM (standing for Backbone Mapping). This study also develops algorithm OOB (standing for Obtaining Optimal Backbones) that exploits a branch-and-bound strategy to obtain the optimal solution efficiently. This study tests the performance of these algorithms on both synthesis and real datasets. Experimental results show that by sharing the intermediate results, the BM and OOB algorithms significantly reduce the total number of messages incurred by multiple aggregate queries, thereby extending the lifetime of sensor networks.

1. Introduction

Recent advances in wireless and embedded technologies have increased the deployment of small and inexpensive wireless sensor nodes in various applications, including field data collection, remote monitoring and control, smart homes, factory automation, and security. In a wireless sensor network, sensor nodes are deployed in a monitored region. Sensor nodes are capable of collecting, processing, and storing environmental information. An access point (i.e., sink) serves as a network interface, issuing queries and collecting readings from sensor nodes. Each node sends its readings to the sink via multi-hop communications at specified sampling periods. Since sensor nodes are usually powered by batteries, energy saving is a vital design issue in wireless sensor networks [16] [1] [41] [14]. For a sensor node, the energy consumption of communication is larger than that of computing and sensing. Thus, minimizing the number of transmitted messages can increase sensor lifetime.

To acquire data from a sensor network, many sensor database management systems, such as MaD-WiSe [2], MauveDB [9], and ADAE [5], support declarative queries. The following example gives a declarative query to obtain the sum of light readings from a set of sensors (S_1,S_2,..., and S_5) every two seconds (i.e., the sampling period is 2 seconds).

* Corresponding author at: No. 1001 University Road, Hsinchu 300, Taiwan. Tel.: +886 3 5731478.
E-mail address: wcpeng@cs.nctu.edu.tw (W.-C. Peng).

0169-023X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2011.03.008

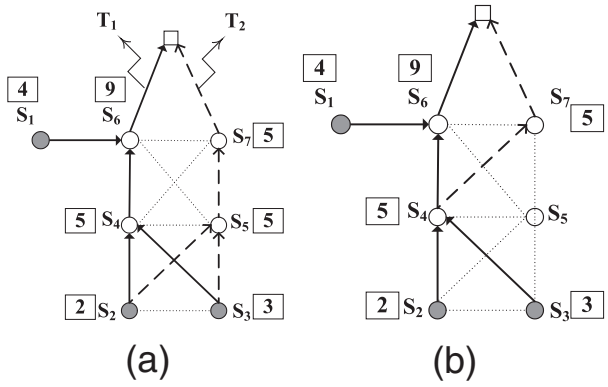

Fig. 1. In-network aggregate queries. (a) Without sharing intermediate results and (b) with sharing intermediate results.

Example 1. A declarative query example:

SELECT SUM(light)
FROM S_1,S_2,S_3,S_4,S_5
SAMPLE PERIOD 2s

Upon receiving these queries, the sink injects them into the wireless sensor network. Routing trees are the most common method of propagating queries and collecting query results from sensor networks, and many routing protocols in various sensor operating systems adopt this approach: the DHV protocol in TinyOS [8], ContikiRPL protocol in Contiki [37], SAMPL protocol in Nano-RK [26], and LiteOS [4]. A routing tree can be viewed as a query tree whose the nodes are sensor nodes that participate in the query processing. The edges between nodes represent the routing paths determined by existing routing protocols. Previous studies propose in-network aggregate query processing, in which sensor nodes use aggregate operators to reduce the number of messages, thereby conserving energy [28][29][43]. When multiple aggregate queries are submitted to wireless sensor networks, it is possible to generate the intermediate results of these queries. Sharing these intermediate results of queries can further reduce the number of messages involved for these queries. The following examples provide some scenarios in which concurrent queries could share some intermediate results:

Example 2. Many recent research projects have deployed sensors in forests to monitor forest environments. To prevent forest fires, forest management units can submit queries to this sensor network to obtain readings of average temperature and relative humidity in the forest. Forest scientists can also use this sensor network to obtain similar readings to analyze environment effects on tree growth. These concurrent queries could share some intermediate results.

Example 3. In a cyber-physical system, there are many sensors deployed in our living environments. Consider a traffic monitoring sensor network application in which sensors are deployed to monitor the average speed of roads. Drivers may query the sensor network to determine road traffic conditions and find the best route. Traffic officers may also query average speeds of some roads to monitor traffic status. Bus corporations utilize these information to predict the time that the next buses will arrive at a station for bus passengers. In this scenario, since users may query traffic speed for the same roads, the system could improve its responsiveness by generating some intermediate results to be shared by all.

Example 4. A recent article reports that U.S. forces deployed sensors in Iraq to protect troops and bring security to towns. Since this sensor network could constantly watch designated areas day and night, many army units may submit long-running queries to detect approaching enemies of the same area. When some enemy armies intrude, sensors in the same area raise alarms such that queries submitted in this area may have common intermediate results. Sharing these intermediate results could deal with queries from many army units efficiently.

To demonstrate the concept of sharing intermediate results, consider the two query trees in Fig. 1, where Q_1 and Q_2 are represented as solid and dashed line, respectively. The data sources of Q_1 (respectively, Q_2) are S_1, S_2, and S_3 (respectively, S_2 and S_3), and sensors nodes S_4, S_5, S_6, and S_7 are relay nodes for these two queries. Assume that the relay nodes perform aggregate operation *SUM*. In Fig. 1, the number associated with a sensor is the intermediate result at that sensor. For example, the intermediate result on sensor S_4 is five (i.e., the sum of intermediate results at S_2 and S_3). In Fig. 1(a), two queries do not share intermediate results. Queries Q_1 and Q_2 have 5 and 4 messages, respectively. Therefore, the total number of messages in Fig. 1(a) is 9. On the other hand, in Fig. 1(b), if Q_1 shares the intermediate result of S_4 with Q_2, sensor S_7 of Q_2 could directly obtain the intermediate result of S_4 without accessing readings at S_2 and S_3 .[1] The total number of messages in Fig. 1(b) is thus 7 (i.e., $5+2$).

[1] Note that we only share intermediate results of sensor nodes, not the data sources. For example, the data sources of sensor S_6 is the intermediate results of sensor S_1 and sensor S_4. Due to the storage constraint of sensors, it is hard to keep all readings of data sources. Thus, we only keep the intermediate result of sensor S_6 (i.e., 9).

Compared to the number of messages incurred in Fig. 1(a) (i.e., 9), the reduced number of messages is 2 (i.e., $9-7=2$). Given a set of query trees, it is possible to further investigate how to share intermediate results among queries. In the example in Fig. 1(b), Q_1 is a backbone and Q_2 is a non-backbone. This figure shows that the backbone is performed as usual and non-backbones must adjust their query trees to access the intermediate results of backbones. Clearly, sharing intermediate results among query trees reduces the number of messages involved in multiple queries.

Given a set of query trees with the same aggregate operation, the proposed approach attempts to determine the sets of backbones and non-backbones, and for each non-backbone, derive the set of backbones that share their intermediate results. Several challenge issues must be overcome to achieve this goal:

(1) Determine the benefit of sharing intermediate results between query trees.
Since there are many possible cases of sharing intermediate results between query trees, it is necessary to determine the benefits of sharing intermediate results. This formulation of benefits should consider the possible sensor nodes for sharing intermediate results and the sampling period of query trees.
(2) Select backbones from a set of query trees.
Given a set of query trees with possible sharing of intermediate results, it is necessary to determine sets of backbones and non-backbones to minimize the number of messages involved. One challenge here is how to determine which query tree should be in the set of backbones. As indicated above, backbones are performed as usual. Thus, the selection of backbones should consider their own query trees and their benefits to other non-backbones. The tree size and sampling period are both important factors in determining the set of backbones.
(3) Determine the set of backbones for each non-backbone.
A non-backbone has many ways to access intermediate results from backbones. Since accessing different intermediate results may reduce the number of messages differently, the set of backbones should be carefully determined for each non-backbone to reduce the maximal number of messages.
(4) Deal with dynamic query workload.
Since queries may join or leave dynamically, it is necessary to develop a mechanism to handle dynamic query workloads.

To address the issues above, this study first formulates a benefit function between query trees based on the sampling period of query trees and possible number of messages reduced. The benefit function makes it possible to model possible cases of sharing intermediate results among query trees as a *reduction graph*, where each vertex is a query tree and the weight between query trees represents the benefit achieved by sharing intermediate results. Based on the devised reduction graph, the determination of backbones and non-backbones is formulated as a Max-Cut problem, in which a cut between two sets (i.e., backbone set and non-backbone set) is derived to maximize the sum of edge weights. Maximizing the sum of edge weights maximizes the number of messages reduced by sharing intermediate results. Since a Max-Cut problem is a NP-hard problem, this study develops a greedy algorithm BM (standing for Backbone Mapping), a heuristic algorithm, to identify the sets of backbones and non-backbones. Since BM may not always obtain the optimal solution, this study further proposes the algorithm OOB (standing for Obtaining Optimal Backbones) to derive the optimal sets of backbones and non-backbones. Note that the OOB algorithm can also evaluate the solution quality derived by the BM algorithm. The OOB algorithm adopts the branch-and-bound strategy. This strategy views the procedure of determining the optimal solution as a search procedure in a state-space tree, where each tree node represents a possible solution for the sets of backbones and non-backbones. This study also develops a bounding function to guide the search procedure to reach the tree node with the optimal solution. A maintenance mechanism incrementally adjusts the set of backbones to handle the dynamic query workloads. This study comparatively analyzes the performance of these algorithms and conducts a sensitivity analysis of several parameters, including the number of queries and the distribution of data sources for queries. Experimental results show that by sharing intermediate results among queries, the BM and OOB algorithms can significantly reduce the number of messages, thereby saving a considerable amount of energy. Furthermore, the solution obtained by the BM algorithm is very close to the optimal one derived by the OOB algorithm, verifying the good solution quality of the BM algorithm. This study also implements a brute-force scheme to derive the optimal solution to justify the design of the bounding function in the OOB algorithm. Compared with the brute-force algorithm, the OOB algorithm can derive the optimal solution with greater efficiency when the number of queries increases.

The remainder of this paper is organized as follows. Section 2 describes related works, while Section 3 presents the preliminaries. Section 4 develops the BM and the OOB algorithms to determine sets of backbones and non-backbones, and their mapping relation. This section also proposes a maintenance mechanism. Section 5 presents performance studies, while Section 6 provides discussions. Section 7 concludes this paper.

2. Related works

Since sensor nodes are usually powered by batteries, energy is a very precious resource for each sensor node. Therefore, minimizing energy consumption is always the most critical issue while the requirements of applications in wireless sensor networks are satisfied. Generally speaking, energy saving could be achieved from several approaches. The first approach is to schedule waking up time and sleeping time in MAC layer [3,21,32]. This approach is usually used in event detection where events are rarely generated. In this case, spending time in the idle period may increase energy consumption of sensors such that sensors would be better in sleep mode as long as possible. The second approach is to design a coding mechanism for reducing the packet size [10,20]. This approach is usually used when the sink is required to receive all the original packets from sensors. The core idea of network coding is to use some operations, such as an XOR or a linear combination, to mix data at intermediate network nodes. Encoding packets at intermediate nodes and then

sending only coded packet instead of individual packets reduces the traffic without increasing delay. Therefore, energy consumption can be reduced by reducing the amount of data transmission. The third approach is to execute in-network processing. The simplest way is in-network aggregation which combines data from different sources by aggregation operations. Based on the fact that the transmission cost is much higher than the computational cost, in-network aggregation can efficiently reduce the number of messages incurred in a wireless sensor network. The main theme of this paper is to explore intermediate results of in-network aggregate queries for further reduce the number of message incurred. Thus, our paper belongs to this category.

The idea of in-network aggregation in sensor networks is first proposed in [28]. The authors of this paper proposed Tiny AGgregation (TAG) for in-network aggregation. Moreover, there are several extensions about in-network aggregation with precision constrained [35,40]. These works dealt with the tradeoff between data quality and energy consumption for long-running query processing. Recently, query optimization for multiple query processing in wireless sensor networks has been addressed. Given a set of query sensors with the same sampling period, the previous work of [36] organized all queried sensors as a dissemination tree and these sensors only report their readings if the readings of these sensors are changed, where the authors in [36] utilized the proposed linear-algebra approach to detect the change of sensor readings. Moreover, the authors in [18] proposed several strategies to pre-compute and store commonly used aggregation results in wireless sensor networks and these results are accessible for multiple queries. The authors in [30] proposed two-tier multiple query optimization in which a cost model is developed in the sink for generating an optimal set of queries and then in-network query optimization is performed. However, queries considered in [30] are not in-network aggregate queries.

Another related research works of this paper discusses query processing and sensor database management systems. Prior works [43] established the concept that a wireless sensor network could be viewed as a database. Then, a SQL-like language is developed for collecting data from wireless sensor networks [29]. Currently, this idea is still widely used for users to submit queries to sensor networks in current sensor database management systems, such as Tailor-made DBMS [25], MaD-WiSe [2], MauveDB [9], and ADAE [5]. Some spatial queries are also developed. For example, K Nearest Neighbor (KNN) query processing are studied in [33,39]. Since sensors may fail or be disrupted by the environment, some papers also focus on detecting or filtering the error readings to guarantee the accuracy of queries [34,42]. Some data collection mechanisms have been studied in literatures [6,15,19]. In wireless ad-hoc network, a multicast tree is proposed to minimize the transmission cost from a given source to a set of receiver [23,27,31]. A multicast tree is similar to the routing tree in wireless sensor networks if the sender is viewed as the sink and the query sensors are viewed as the receiver. Generally, the solution of finding multicast trees are based on finding Steiner trees, shortest path tree, and so on. However, different from this paper, most studies of multicast trees concentrate on how to construct a multicast tree with the minimum communication cost or minimum data-overhead. In this paper, we study how to share the intermediate results among multiple query trees.

In this paper, we not only consider the sampling period of aggregate queries but also the dynamic workloads of wireless sensor networks into the problem formulation. Our heuristic algorithm BM not only determines the sets of backbones and non-backbones but also derives mapping relationships between backbones and non-backbones. To obtain the optimal solution, we further propose algorithm OOB. Furthermore, we develop a maintenance mechanism to deal with the dynamic workload in wireless sensor networks. Furthermore, an extensive performance study is conducted in both synthetic and real datasets, and sensitivity analysis is investigated. To the best of our knowledge, no previous study has exploited the query optimization for in-network aggregate queries, let alone devising algorithms to determine the backbone set and the non-backbone set for sharing intermediate results. These features distinguish this paper from others.

3. Preliminaries

Section 3.1 presents some notations and the problem formulation. Then, Section 3.2 develops a reduction graph, which is used to determine benefits of sharing intermediate results among queries.

3.1. Problem formulation

After receiving queries, the sink propagates them via existing routing protocols and collects the corresponding query trees. Using this set of query trees, the goal of the proposed approach is to share intermediate results among query trees to minimize the number of messages. To facilitate the presentation of our paper, some notations are defined (Table 1). Each query is represented as a query tree, expressed by T_i, where i is the identification of queries. As indicated above, each query T_i is periodically performed every $E(T_i)$ time units, where $E(T_i)$ is the sampling period for query T_i. When performing query T_i each time, the number of messages incurred is expressed as $N(T_i)$, the number of edges in T_i. Given a set of queries, denoted as Q, the goal is to determine sets of backbones and non-backbones, where each non-backbone is able to access intermediate results from backbones to reduce the total number of messages incurred. The set of backbones (respectively, non-backbones) is represented as B (respectively, NB). Note that a non-backbone is able to access intermediate results from multiple backbones. Thus, the set of backbones that share intermediate results for non-backbone T_i is defined as $B(T_i)$. Given a non-backbone T_i and its backbone T_j, suppose that a sensor $S_m \in T_i$ accesses the intermediate results from $S_n \in T_j$. The number of messages reduced under the scenario in which S_m obtains the intermediate results of S_n is expressed as $R_{i,j}(S_m, S_n)$. Note that (S_m, S_n) is called a *sharing pair* of T_i and T_j. There are many possible ways for a non-backbone to access the intermediate results of its backbone. The sections below we will discuss how to derive the sharing pairs that can maximize the number of messages reduced.

Table 1
Description of notations.

Description	Symbol
Query tree of Q_i	T_i
Number of messages associated with T_i	$N(T_i)$
Sampling period of T_i	$E(T_i)$
Backbone set	B
Non-backbone set	NB
The set of backbones tree of T_i	$B(T_i)$
Number of messages reduced under the scenario that $S_m \in T_i$ accesses intermediate results of S_n in $T_j \in B(T_i)$	$R_{i,j}(S_m,S_n)$
Probability that T_i can access the intermediate results from $T_j \in B(T_i)$	$P(T_i,T_j)$
Weight of an edge (v_i,v_j) in a reduction graph	$w(v_i,v_j)$

Example 5. Assume that in Fig. 2, T_1 is a backbone and T_2 is a non-backbone. Fig. 3(a) shows that by accessing the intermediate result at S_{11} in T_1, $R_{2,1}(S_{12},S_{11})=4$. As Fig. 3(a) shows, to access intermediate results at S_{11} in T_1, query tree T_2 must adjust its corresponding tree. An alternative way to access the intermediate results of T_1 is to obtain the intermediate results at S_9, as Fig. 3(b) shows. Compared to the case in Fig. 3(a), the value of $R_{2,1}(S_{10},S_9)$ (i.e., 1) is not maximum. Therefore, the sharing pair (S_{12},S_{11}) would be better than (S_{10},S_9) in terms of reducing the number of messages.

As indicated above, each query has its own sampling period. Clearly, non-backbones can only access intermediate results of backbones when backbones are performed at their sampling time periods. Given one non-backbone T_i and one backbone T_j, for T_i,

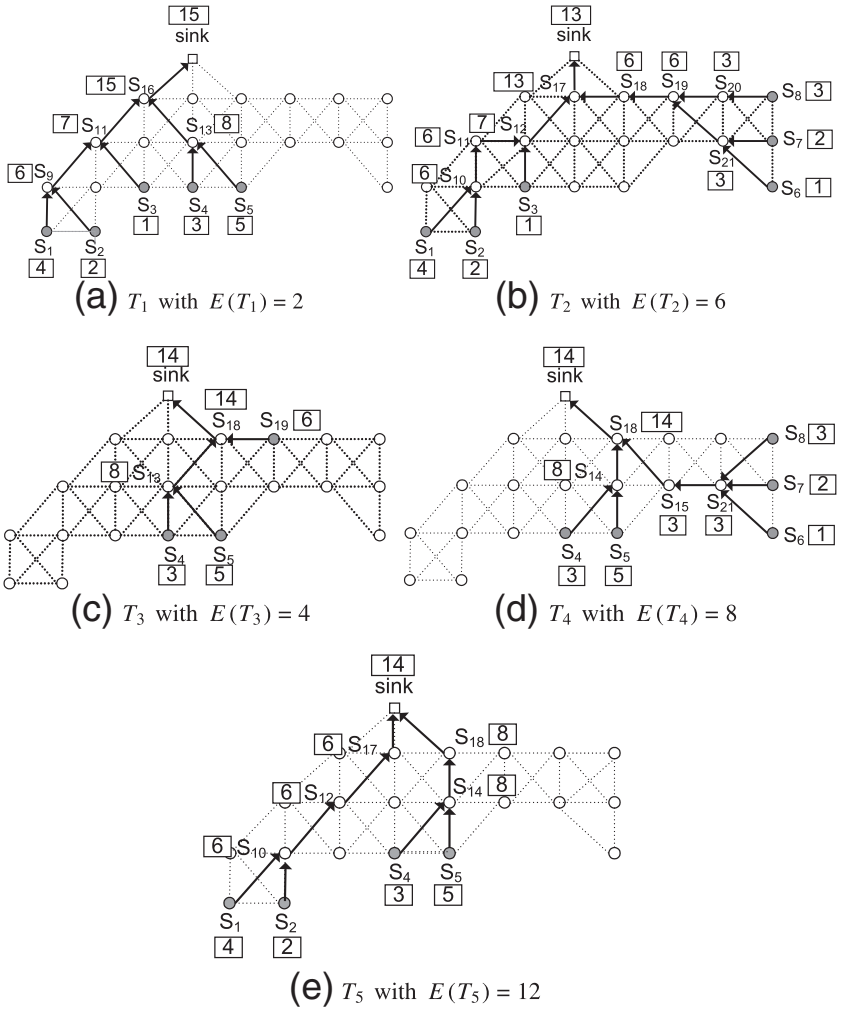

Fig. 2. Five query trees with the aggregate operation SUM and their sampling periods.

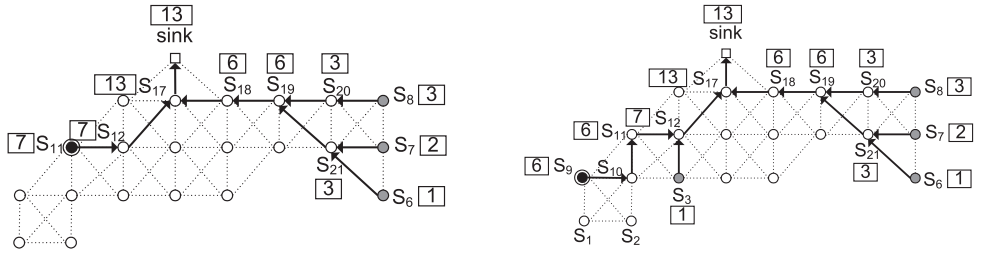

Fig. 3. Adjusting the query tree T_2 for the intermediate results of T_1 at black sensors with circles.

derive the probability of accessing intermediate results of T_j, denoted as $P(T_i,T_j)$. Both T_i and T_j are performed every $lcm(E(T_i),E(T_j))$ time units, where $lcm(\cdot)$ is the largest common multiplier of $E(T_i)$ and $E(T_j)$. For every $E(T_i)$, non-backbone T_i is executed. Based on this observation, the value of $P(T_i,T_j)$ is $P(T_i,T_j) = \frac{E(T_i)}{lcm(E(T_i),E(T_j))}$.

Example 6. Consider the query trees in Fig. 2. Fig. 4 shows their execution schedule according to their sampling time periods. Suppose that T_4 is a backbone for non-backbone T_2. Note that T_2 only could access intermediate results of T_4 at the 24th time unit. For other time slots, T_2 must still perform its own query tree as usual. Therefore, the probability of non-backbone T_2 accessing intermediate results of T_4 is $\frac{6}{lcm(6,8)} = \frac{1}{4}$.

Similar to [30], queries are submitted or left dynamically. In this dynamic environment, a cost model is used to estimate the total number of messages incurred. The cost model derived is the average number of messages incurred per time unit, denoted as *AvgCost*. Assume that the number of queries, their execution time, and the maximal time monitor length are given. The total number of messages incurred is proportional to the value of *AvgCost*. To derive *AvgCost*, consider the average number of messages incurred in sets of backbones and non-backbones. Denote $AvgCost_B(T_j)$ as the average number of messages per time unit for backbone T_j, and $AvgCost_{NB}(T_i,T_j)$ as the average number of messages per time unit for non-backbone T_i by accessing the intermediate results of backbone T_j. A set of queries Q with the sets of backbone B and non-backbone NB leads to following formula:

$$AvgCost(Q) = \sum_{T_j \in B} AvgCost_B\left(T_j\right) + \sum_{T_i \in NB} \sum_{T_j \in B(T_i)} AvgCost_{NB}\left(T_i,T_j\right). \tag{1}$$

For a backbone T_j, the number of messages incurred is $N(T_j)$ and the time period for executing T_j is every $E(T_j)$ time units. Thus, the average number of messages incurred by T_j is $AvgCost_B(T_j) = \frac{N(T_j)}{E(T_j)}$. Next, derive the average number of messages incurred by $S_m \in T_i$ using the share of intermediate results of $S_n \in T_j$. As indicated above, both backbone and non-backbone are periodically executed according to their sampling time periods. For non-backbone T_i, the probability of accessing intermediate results of T_j is $P(T_i,T_j)$. Assume that T_i accesses its backbone T_j via a sharing pair (S_m,S_n). Thus, the number of messages incurred is $P(T_i,T_j)\times(N(T_i) - R_{i,j}(S_m,S_n))$. On the other hand, the probability of performing T_i is $(1-P(T_i,T_j))$. When T_i cannot access its backbone T_j, the corresponding number of messages is as $(1-P(T_i,T_j))\times N(T_i)$. If the set of sharing pairs of T_i and T_j is $\chi_{i,j}$, then $AvgCost_{NB}(T_i,T_j)$ can be formulated as follows:

$$AvgCost_{NB}\left(T_i,T_j\right) = \sum_{(S_m,S_n)\in\chi_{i,j}} \frac{\left(1-P\left(T_i,T_j\right)\right)\times N(T_i) + P\left(T_i,T_j\right)\times\left(N(T_i)-R_{i,j}(S_m,S_n)\right)}{E\left(T_j\right)}. \tag{2}$$

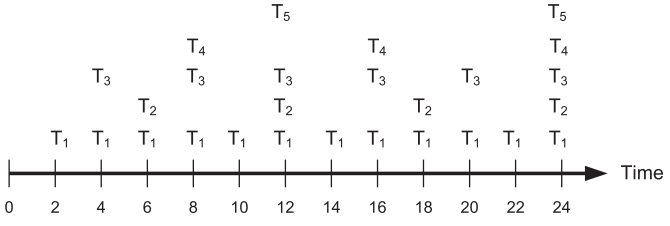

Fig. 4. Query execution schedule.

The formula for *AvgCost(Q)* leads to the following derivations:

$$
\begin{aligned}
&AvgCost(Q)\\
&= \sum_{T_j \in B} AvgCost_B\left(T_j\right) + \sum_{T_i \in NB} \sum_{T_j \in B(T_i)} AvgCost_{NB}\left(T_i, T_j\right)\\
&= \sum_{T_j \in B} \frac{N\left(T_j\right)}{E\left(T_j\right)} + \sum_{T_i \in NB} \sum_{T_j \in B(T_i)} \sum_{(S_m,S_n) \in \chi_{i,j}} \frac{\left(1-P\left(T_i,T_j\right)\right) \times N(T_i) + P\left(T_i,T_j\right) \times \left(N(T_i) - R_{i,j}(S_m,S_n)\right)}{E(T_i)}\\
&= \sum_{T_j \in B} \frac{N\left(T_j\right)}{E\left(T_j\right)} + \sum_{T_i \in NB} \sum_{T_j \in B(T_i)} \sum_{(S_m,S_n) \in \chi_{i,j}} \frac{N(T_i) - P\left(T_i,T_j\right) \times R_{i,j}(S_m,S_n)}{E(T_i)}\\
&= \left(\sum_{T_j \in B} \frac{N\left(T_j\right)}{E\left(T_j\right)} + \sum_{T_i \in NB} \frac{N(T_i)}{E(T_i)}\right) - \sum_{T_i \in NB} \sum_{T_j \in B(T_i)} \sum_{(S_m,S_n) \in \chi_{i,j}} \frac{P\left(T_i,T_j\right) \times R_{i,j}(S_m,S_n)}{E(T_i)}\\
&= \sum_{T_k \in (B \cup NB)} \frac{N(T_k)}{E(T_k)} - \sum_{T_i \in NB} \sum_{T_j \in B(T_i)} \sum_{(S_m,S_n) \in \chi_{i,j}} \frac{\frac{E(T_i)}{lcm\left(E(T_i), E\left(T_j\right)\right)} \times R_{i,j}(S_m,S_n)}{E(T_i)}\\
&= \sum_{T_k \in (B \cup NB)} \frac{N(T_k)}{E(T_k)} - \sum_{T_i \in NB} \sum_{T_j \in B(T_i)} \sum_{(S_m,S_n) \in \chi_{i,j}} \frac{R_{i,j}(S_m,S_n)}{lcm\left(E(T_i), E\left(T_j\right)\right)}.
\end{aligned}
$$

Consequently,

$$
AvgCost(Q) = \sum_{T_k \in (B \cup NB)} \frac{N(T_k)}{E(T_k)} - \sum_{T_i \in NB} \sum_{T_j \in B(T_i)} \sum_{(S_m,S_n) \in \chi_{i,j}} \frac{R_{i,j}(S_m,S_n)}{lcm\left(E(T_i), E\left(T_j\right)\right)}. \tag{3}
$$

With the above derivations, it is possible to minimize *AvgCost(Q)* by maximizing the sum of $\sum_{(S_m,S_n) \in \chi_{i,j}} \frac{R_{i,j}(S_m,S_n)}{lcm(E(T_i),E(T_j))}$, where $T_j \in B(T_i)$. Since $\frac{R_{i,j}(S_m,S_n)}{lcm(E(T_i),E(T_j))}$ represents the number of messages reduced by a sharing pair (S_m,S_n), the benefit of a sharing pair (S_m,S_n) can be defined as $benefit_{i,j}(S_m,S_n) = \frac{R_{i,j}(S_m,S_n)}{lcm\left(E(T_i),E\left(T_j\right)\right)}$. Accordingly, the problem addressed in this paper is formally described as follows:

Problem. Given a set of query trees *Q*, determine: 1) sets of backbones *B* and non-backbones *NB*, and 2) the corresponding set of backbones for each non-backbone. The objective of this problem is to maximize the sum of benefits of sharing the intermediate results of backbones.

The benefit formula derived above can judiciously determine the set of backbones and the sharing pairs. Clearly, the backbones selected should maximize the number of message reduced for non-backbones. On the other hand, the sampling periods of queries should be taken into account in selecting backbones. If the sampling period of backbones are too large, non-backbones will have greater difficulty obtaining the intermediate results of backbones. However, a smaller sampling period of backbones will increase the number of messages incurred during the execution of backbones. Thus, backbone selection must strike a compromise between the number of messages reduced and the sampling period of queries. Once the set of backbones is determined, it is possible to determine a set of backbones and the sharing pairs for each non-backbone.

3.2. Determination of benefits among query trees

Given a set of query trees, this section derives a reduction graph that captures the amount of benefits between query trees. The reduction graph is defined as follows:

Definition 1. *Reduction Graph:* Given a set of query trees *Q*, a reduction graph is a weighted directed graph $G=(V,E)$. A vertex $v_i \in V$ represents a query tree $T_i \in Q$. For each sharing pair (S_m,S_n) of T_i and T_j, there is an edge $(v_i,v_j) \in E$ with its label as the data source of (S_m,S_n) and its weight as $max\{0, benefit_{i,j}(S_m,S_n)\}$.

One challenge to building a reduction graph is to determine the labels and weights of edges. As Section 3.1 indicates, given a sharing pair (S_m,S_n) of two query trees T_i and T_j, $benefit_{i,j}(S_m,S_n)$ is modeled as $\frac{R_{i,j}(S_m,S_n)}{lcm(E(T_i),E(T_j))}$. The value of $lcm(E(T_i),E(T_j))$ is straightforwardly determined according to the sampling periods of query trees. The weights of edges should be as large as possible to maximize the sum of benefits. Therefore, it is necessary to derive the sharing pairs (S_m,S_n) that maximize the number of messages reduced.

The value of $R_{i,j}(S_m,S_n)$ is related to the sharing pair (S_m,S_n) of T_i and T_j. In other words, S_m of T_i gets the intermediate result from S_n of T_j. Clearly, we could save the number of messages required for S_m of T_i, denoted as $N_i(m)$. However, an extra admission cost is

required to access the intermediate result at S_n from S_m. Denote as $d_S(m,n)$, this cost is estimated as the minimum hop counts between S_m and S_n. Therefore, the number of messages reduced is $N_i(m) - d_S(m,n)$.

The scenario above shows an example case in which S_m of T_i accesses the intermediate result at S_n of T_j. In reality, there are many possible sharing pairs between two query trees. For ease of presentation, each case uses the sharing pair as its identification. To explore possible cases of sharing pairs, it is necessary to check the data sources of all sensors in these two query trees. Given two query trees T_i and T_j, if the data sources for one sensor in T_i are the same as those for another sensor in T_j, these two sensors form a sharing pair. It is possible to further determine the number of messages reduced by this sharing pair (i.e., $R_{i,j}(S_m,S_n)$). However, many sharing pairs may be generated if their data sources are the same. Note that since query trees are hierarchical, sharing intermediate results at higher level sensor nodes can eliminate more messages. Thus, it is not necessary to enumerate all sharing pairs. Sharing pairs with the maximal set of data sources between query trees should be generated. Consider T_1 and T_2 as an example, where T_2 intends to access intermediate results of T_1. Fig. 2 shows that $S_9 \in T_1$ and $S_{10} \in T_2$ have the same intermediate results because they have the same set of data sources (i.e., $\{S_1,S_2\}$). However, the sharing pair (S_9,S_{10}) is not the sharing pair with the maximal set of data sources because the sharing pair (S_{11},S_{12}) has the data sources $\{S_1,S_2,S_3\}$, which contain the set of data sources of the sharing pair (S_9,S_{10}). The value of $R_{i,j}(S_m,S_n)$ is the maximum number of messages reduced among the set of sharing pairs that have the maximal set of data sources.

Example 7. Table 2 shows the corresponding numbers of messages reduced for the query trees in Fig. 2. Fig. 5 depicts the reduction graph for these query trees. For example, the first row of the first block in Table 2 shows that the data source of the sharing pair (S_{11},S_{12}) of T_1 and T_2 is $\{S_1,S_2,S_3\}$, and $R_{1,2}(S_{11},S_{12})=3$. Therefore, the edge (v_1,v_2) is labeled as $\{S_1,S_2,S_3\}$, and the weight of this edge is $benefit_{1,2} = (S_{11},S_{12}) = \frac{R_{1,2}(S_{11},S_{12})}{|cm(E(T_1),E(T_2))|} = \frac{3}{6} = \frac{1}{2}$. Consider another example of the sharing pair (S_{18},S_{15}) of T_2 and T_4. The second row of the second block in Table 2 shows that the data source of this pair is $\{S_7,S_8,S_9\}$ and $R_{2,4}(S_{18},S_{15})=5$. Therefore, the edge (v_2,v_4) is labeled $\{S_7,S_8,S_9\}$ and the weight of (v_2,v_4) is $benefit_{2,4}(S_{18},S_{15}) = \frac{R_{2,4}(S_{18},S_{15})}{|cm(E(T_2),E(T_4))|} = \frac{5}{24}$.

4. Query optimization by selecting backbone trees

This section develops two algorithms to determine the set of backbones and the mapping between the backbone set and the non-backbone set. Section 1 develops a heuristic algorithm BM (standing for Backbone Mapping). To obtain the optimal solution, Section 4.2 develops algorithm OOB (standing for Obtaining Optimal Backbones), which is based on a branch-and-bound strategy. Since queries may join or leave dynamically, Section 4.3 proposes a maintenance mechanism.

4.1. Algorithm BM

Using the reduction graph, the BM algorithm determines the set of backbones and mapping relationships among backbones and non-backbones. the BM algorithm consists of two phases: the *partition phase* and the *mapping phase*. The partition phase determines sets of backbones and non-backbones with the purpose of maximizing the sum of benefits between the backbone set and the non-backbone set. The mapping phase obtains the corresponding set of backbones for each non-backbone.

Table 2
The sharing pairs of query trees.

	Data source	Value
$R_{1,2}(S_{11},S_{12})$	$\{S_1,S_2,S_3\}$	3
$R_{1,3}(S_{13},S_{13})$	$\{S_4,S_5\}$	2
$R_{1,4}(S_{13},S_{14})$	$\{S_4,S_5\}$	1
$R_{1,5}(S_9,S_{10})$	$\{S_1,S_2\}$	1
$R_{1,5}(S_{14},S_{13})$	$\{S_4,S_5\}$	1
$R_{2,1}(S_{12},S_{11})$	$\{S_1,S_2,S_3\}$	4
$R_{2,4}(S_{18},S_{15})$	$\{S_7,S_8,S_9\}$	5
$R_{2,5}(S_{10},S_{10})$	$\{S_1,S_2\}$	2
$R_{3,1}(S_{13},S_{13})$	$\{S_4,S_5\}$	2
$R_{3,4}(S_{13},S_{14})$	$\{S_4,S_5\}$	1
$R_{3,5}(S_{13},S_{14})$	$\{S_4,S_5\}$	1
$R_{4,1}(S_{14},S_{13})$	$\{S_4,S_5\}$	1
$R_{4,2}(S_{15},S_{19})$	$\{S_7,S_8,S_9\}$	3
$R_{4,3}(S_{14},S_{13})$	$\{S_4,S_5\}$	1
$R_{4,5}(S_{14},S_{14})$	$\{S_4,S_5\}$	2
$R_{5,1}(S_{10},S_9)$	$\{S_1,S_2\}$	1
$R_{5,1}(S_{14},S_{13})$	$\{S_4,S_5\}$	1
$R_{5,2}(S_{10},S_{10})$	$\{S_1,S_2\}$	2
$R_{5,3}(S_{14},S_{13})$	$\{S_4,S_5\}$	1
$R_{5,4}(S_{14},S_{14})$	$\{S_4,S_5\}$	2

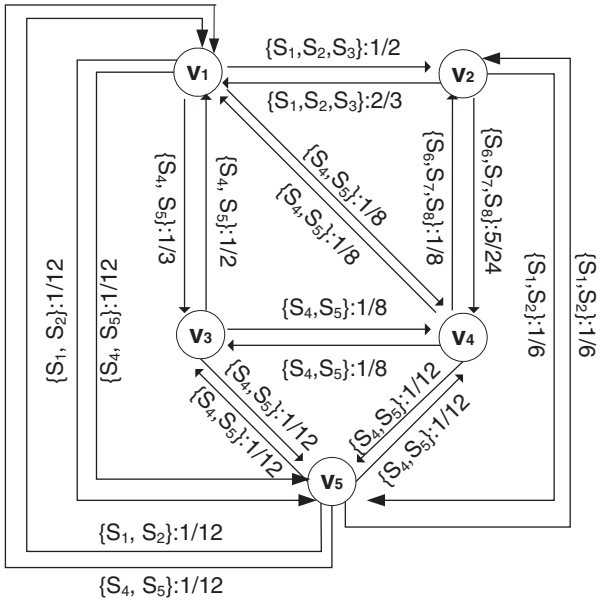

Fig. 5. The reduction graph for five query trees, where each edge is associated with its label and weight marked as *label:weight*.

4.1.1. Partition phase

Given a reduction graph, it is possible to determine the backbone set and the non-backbone set that maximize the sum of edge weights both backbone and non-backbone sets. Clearly, maximizing the sum of edge weights among the backbone set and the non-backbone set could maximize the number of messages reduced via the intermediate results of backbones. The problem can be modeled as a Max-Cut problem. The input of a Max-Cut problem is a graph $G=(V,E)$, where V is the set of vertices and E is the set of edges with corresponding weights. A Max-Cut problem divides the vertices into two sets with the goal of maximizing the sum of edge weights between these two sets. Given a reduction graph, a maximum cut is a sum of edge weights in which the two vertices of each edge belong to two sets (e.g., B and NB), such that $B\cup NB=V$, $B\cap NB=\phi$ and the sum of edge weights is maximized. Since a Max-Cut problem is a NP-hard problem, this study proposes a heuristic problem to determine a Max-Cut from the reduction graph. Note that the backbone set and the non-backbone set are determined after a maximum cut is derived.

Same as in most papers for solving a Max-Cut problem, a vertex moves from one set to the other set if the movement can bring some benefits. The benefit here indicates how much reduction a tree can achieve when it becomes a backbone. Following this concept, set the backbone set as empty and the non-backbone set as the set of all vertices. For a vertex in the non-backbone set, determine the corresponding *backbone gain* and use the gain to estimate the benefit when this vertex is selected as a backbone.

Definition 2. *Backbone Gain:* Given a reduction graph, the backbone gain achieved by selecting v_i as a backbone, denoted by $\delta(v_i)$, is

$$\delta(v_i)=\sum_{v_j\in NB-\{v_i\}}\ \sum_{v_k\in B\cup\{v_i\}} w\left(v_j,v_k\right)-\sum_{v_j\in NB}\ \sum_{v_k\in B} w\left(v_j,v_k\right). \tag{4}$$

For each vertex in the non-backbone set, the corresponding backbone should be calculated. The vertex with the maximal backbone gain is then selected in the backbone set. After selecting one query tree as a backbone, the backbone gains for query trees in the non-backbone set are updated. Similarly, according to the backbone gains of query trees in the non-backbone set, the one

Table 3
An execution scenario in the partition phase of the BM algorithm.

Selection iteration	Backbone and non-backbone set	Backbone gain				
		v_1	v_2	v_3	v_4	v_5
1	$B=\{\},NB=\{v_1,v_2,v_3,v_4,v_5\}$	$\frac{35}{24}$*	$\frac{19}{24}$	$\frac{13}{24}$	$\frac{11}{24}$	$\frac{1}{2}$
2	$B=\{v_1\},NB=\{v_2,v_3,v_4,v_5\}$	–	$-\frac{3}{8}$	$-\frac{7}{24}$	$\frac{7}{24}$*	$\frac{1}{6}$
3	$B=\{v_1,v_4\},NB=\{v_2,v_3,v_5\}$	–	$\frac{3}{4}$	$\frac{11}{12}$	–	$\frac{35}{24}$*
4	$B=\{v_1,v_5,v_4\},NB=\{v_2,v_3\}$	–	$-\frac{11}{12}$	$-\frac{5}{12}$	–	–

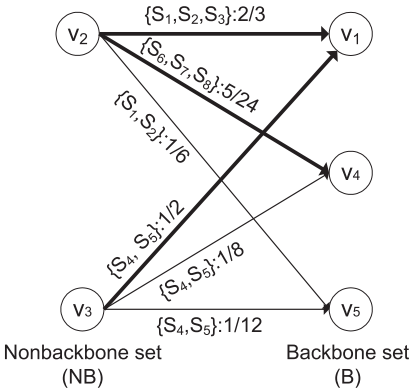

Fig. 6. The mapping between *NB* and *B*.

with the maximal backbone gain is selected as a backbone. The BM algorithm moves query trees from the non-backbone set into the backbone set iteratively until query trees in the non-backbone set have backbone gains smaller than zero. This process divides a set of query trees into backbone and non-backbone sets, and maximizes the cut between the backbone set and the non-backbone set, which results in the maximal sum of benefits.

Example 8. Consider the five query trees in Fig. 2, whose the corresponding reduction graph appears in Fig. 5. Table 3 lists the execution scenario of the partition phase in the BM algorithm. The backbone gains of these query trees are initially calculated. For example, $\delta(v_2) = w(v_1,v_2) + w(v_3,v_2) + w(v_4,v_2) + w(v_5,v_2) - w(v_1,\phi) - w(v_2,\phi) - w(v_3,\phi) - w(v_4,\phi) - w(v_5,\phi) = \frac{1}{2} + 0 + \frac{1}{8} + \frac{1}{6} - 0 - 0 - 0 - 0 = \frac{19}{24} > 0$. Note that the maximal backbone gain is marked with an asterisk. In the first iteration, v_1 is selected into the backbone set since v_1 brings the maximal benefit to other query trees in the non-backbone set. The backbone gains of non-backbone query trees are then updated. In the second iteration, v_4 is included in the backbone set, and all vertices in the non-backbone set update their corresponding backbone gains. Similarly, v_5 is selected into the backbone set. Finally, since all vertices in the non-backbone set have backbone gains smaller than zero, the BM algorithm terminates. This results in the backbone set $B = \{v_1, v_4, v_5\}$ and the non-backbone set $NB = \{v_2, v_3\}$. Fig. 6 shows that this produces a maximum cut as $\frac{2}{3} + \frac{1}{2} + \frac{5}{24} + \frac{1}{8} + \frac{1}{6} + \frac{1}{12} = \frac{7}{4}$. The maximum cut in the example above indicates the maximum value of benefits can be achieved if non-backbones can access the intermediate results of backbones.

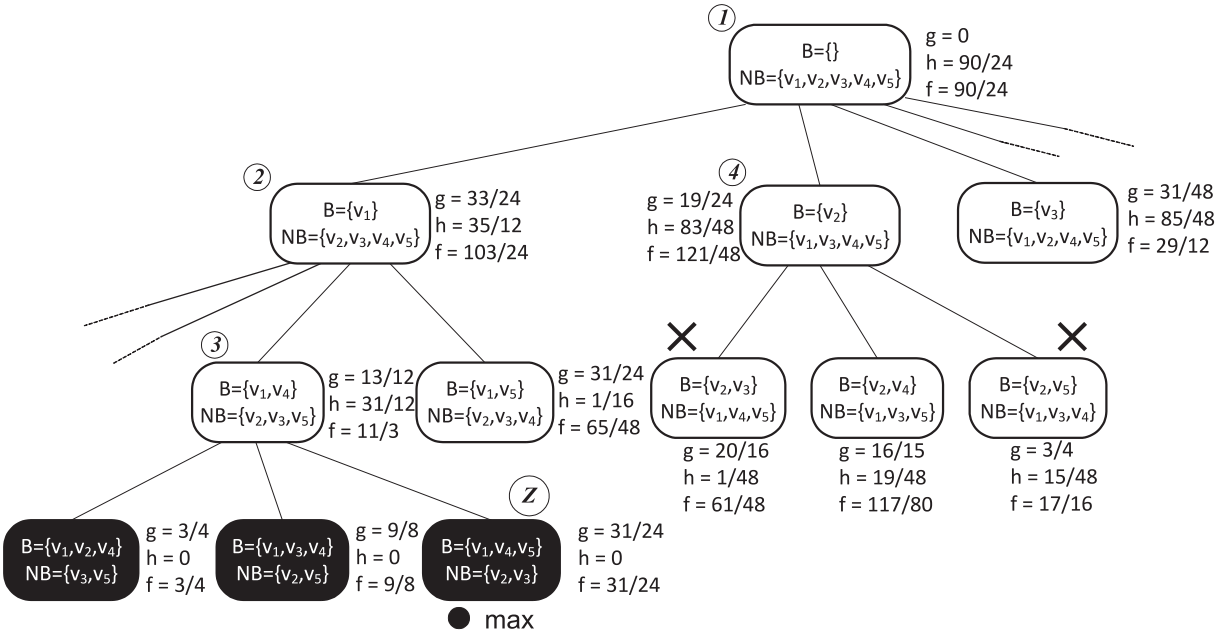

Fig. 7. Snapshot of the state space tree. Black nodes represent nodes that do not needed to expand. Nodes with X notation represent pruned nodes.

4.1.2. Mapping phase

The partition phase maximizes the sum of edge weights between sets of backbones and non-backbones. Next, the mapping phase selects the backbones for each non-backbone. This process can be viewed as mapping relationships from a non-backbone to a set of backbones. A set of backbones for each non-backbone should be carefully selected since the benefit will be over-estimated if a non-backbone selects backbones with overlapping sources. For example, Fig. 6 shows that a non-backbone v_2 has three edges that are linked to all backbones. Among these three edges, the label set of edge (v_2, v_1) and edge (v_2, v_5) have some common sources (i.e., s_1 and s_2). Clearly, if non-backbone v_2 selects v_1 as its backbone (i.e., edge (v_2, v_1) is selected), there is no need to select v_5 since the edge (v_2, v_1) includes the intermediate result from sensors s_1 and s_2. Therefore, a non-backbone should select backbones with disjoint sensors to maximize the sum of benefits.

The problem in this phase is to select a set of edges from the non-backbone set to the backbone set into a mapping set M such that the maximal sum of edge weights in M and the labels of edges in M for each non-backbone are disjoint. To maximize the sum of edge weights in M, select an edge with larger weight, and specifically, an edge whose label set contains more sensors. Note that we should include more sensors that are not in the union set of labels of edges in M for the constraint of disjoint. Given a maximal cut in the reduction graph, the edges between the backbone set and the non-backbone set are assigned to their corresponding scores.

Definition 3. *Scoring Function:* Assume that v_i is a non-backbone and the set of selected edges is M_i. Let the label of an edge (v_i, v_j) be $L(v_i, v_j)$ and the weight of an edge (v_i, v_j) be $w(v_i, v_j)$. The scoring function of an edge (v_i, v_j) is

$$s\left(v_i, v_j\right) = w\left(v_i, v_j\right) \times \left|L\left(v_i, v_j\right) - \bigcup_{(v_i, v_k) \in M_i} L(v_i, v_k)\right|. \tag{5}$$

The philosophy of this scoring function is to evaluate how many weights a mapping relation can obtain if the edge (v_i, v_j) is selected. Since the weight of an edge represents the benefit a non-backbone can obtain by accessing the intermediate results from a backbone, the edge with a larger weight should be added to a mapping relation to maximize its weight. This factor reflects on the first term of the scoring function. On the other hand, a non-backbone is likely to reduce more messages if it can share more intermediate results from different backbones. Thus, a non-backbone should select data sources that do not overlap with the data sources of the current mapping relation. The second term of the scoring function reflects this requirement. The scoring function above associates the set of edges from v_i in the non-backbone set to backbones with a score. One should select the edge with the maximal score and its label set not containing any common sensors in the union set of label sets of edges in M_i. Once an edge is selected in M_i, update the scores of other edges incident to v_i. If the label set of a edge overlaps with the union of label sets in M_i, this edge will be discarded. Repeat the operations above until no edge can be selected. In this case, M_i is a mapping for a non-backbone v_i such that the label set of edges in M_i are disjoint.

Example 9. Consider the example in Fig. 5, where $v_2 \in NB$, the scores of all incident edges are $s(v_2, v_1) = \frac{2}{3} \times 3 = 2$, $s(v_2, v_4) = \frac{5}{24} \times 3 = \frac{5}{8}$, and $s(v_2, v_5) = \frac{1}{6} \times 2 = \frac{1}{3}$. In the beginning, the edge (v_2, v_1) is selected into M_2. The scores of the edges are updated accordingly. Thus, $s(v_2, v_4) = \frac{5}{24} \times 3 = \frac{5}{8}$, and $s(v_2, v_5) = \frac{1}{6} \times 0 = 0$ because $L(v_2, v_5) = \{S_1, S_2\}$. Therefore, the edge (v_2, v_4) is selected into M_2. Since there is no edge incident to v_2 with the label set disjoint to $\{S_1, S_2, S_3, S_7, S_8, S_9\}$, we do not select other edges into M_2. Following the same operations, derive the mapping set $M_3 = \{(v_3, v_1)\}$. Fig. 6 depicts this mapping relationships, where the bold lines are the edges selected for each non-backbone. Finally, $M_2 = \{(v_2, v_1), (v_2, v_4)\}$ and $M_3 = \{(v_3, v_1)\}$. Consequently, the set of backbones for T_2 (respectively, T_3) is $\{T_2, T_4\}$ (respectively, $\{T_1\}$). The sum of the edges for the mapping relation is $\frac{11}{8}$.

Algorithm 1. BM: Backbone Mapping
Input: A reduction graph $G = (V, E)$
Output: Mapping relation M

```
1: /* Partition Phase */
2: NB ← V;
3: B ← φ;
4: while it exists some query tree whose backbone gain is larger than zero do
5:     Select v_i as backbone whose backbone gain is maximal among all query trees in NB;
6:     B ← B ∪ {v_i};
7:     NB ← NB − {v_i};
8: /* Mapping Phase */
9: M ← φ;
10: for each u_i ∈ NB do
11:     while there exists an edge (u_i, v_k) such that L(u_i, v_k) ∩ ∪_{(u_i, v_j) ∈ M} L(u_i, v_j) ≠ φ do
12:         v_j ← the vertex in B with the maximum s(u_i, v_j);
13:         add (u_i, v_j) into M;
```

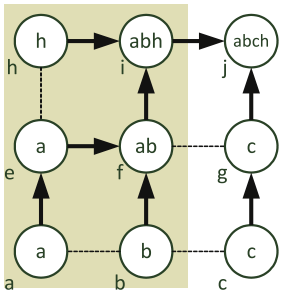

Fig. 8. An illustrative example of generating a sharing region.

Analysis. The time complexity of algorithm BM is analyzed. The backbone gains of all non-backbones should be updated for each iteration of backbone selection. For each iteration in the partition phase, each non-backbone query tree T_j should compute $\delta(T_j)$. Suppose that there are i vertices in NB and $|V|-i$ vertices in B in one iteration. Thus, computing $\delta(T_j)$ for a vertex requires at most $(|V|-i)$ iterations to summarize the edge weights from the reduction graph. Therefore, in the worst case, the time complexity in the partition phase is $\sum_{i=1}^{|V|}(|V|-i)=O(|V|^2)$. In the mapping phase, the time complexity for each vertex in NB to pick the vertex which induces the edge with maximum score is at most $O(|V|)$. Therefore, $O(|V|^2)$ is an asymptotic upper bound of this phase. Hence, the BM algorithm needs at most $O(|V|^2)$.

4.2. Algorithm OOB: obtaining optimal backbones

To obtain the optimal solution, this study proposes algorithm OOB (standing for obtaining optimal backbones), which exploits the branch-and-bound strategy [24] to derive the optimal backbone set. After deriving the optimal backbone set, other query trees not in the optimal backbone set are thus put in the non-backbone set. Then, one could perform the operation in the mapping phase of the BM algorithm to derive the mapping relationships between the backbone set and the non-backbone set. Determining optimal backbones is a search problem on a state space tree, where each node is associated with a state of backbones. Fig. 7 shows part of an example state space tree that corresponds to the case of selecting backbones among the query trees in Fig. 2. Node P in the state space tree contains a possible solution that includes the backbone set, denoted as $P.B$, and the non-backbone set, expressed as $P.NB$. The goal node contains the solution of the optimal backbone set that minimizes the expected number of messages in all query trees. The search for the goal node starts from the root node of the state space tree. Clearly, if one node is in level i of the state space tree, this node has $i-1$ backbones. Thus, the root node will expand its child nodes to search for possible solutions that have one backbone. This procedure of discovering child nodes (referring to the branch operation) continues until the goal node is reached.

Note that the challenge here is to branch nodes that are likely to reach the goal node. The branch operation in the OOB algorithm is controlled by a bounding function $f(*)$ that estimates the upper bound of the sum of benefit in the current mapping of the node P. The $f(P)$ consists of two components: (1) the total benefit achieved by the current P.B and P.NB (referring to as $g(P)$), and (2) the expected cost of arriving at the *goal node* from node P (referring to the $h(P)$). As a result, for each node P, $f(P)$ equals to $g(P)+h(P)$ (i.e., $f(P)=g(P)+h(P)$). The value of $g(P)$ is the total benefit between the backbone set P.B and the non-backbone set P.NB, and is determined by the mapping phase of the BM algorithm. On the other hand, function $h(P)$ is the estimated cost of moving from node P to the goal node. Selecting those query trees for which $\delta(T_j)$ is larger than zero. Thus, $h(P)$ estimates the sum of all non-backbones whose backbone gains are larger than zero to ensure that the goal node is reached. Hence, $h(P)$ is formulated as $h(P)=\sum_{v_j\in P.NB} max(\delta(v_j),0)$.

Example 10. Consider the node Z with $B=\{v_1,v_4,v_5\}$ and $NB=\{v_2,v_3\}$ in Fig. 7. This case is exactly the same as the example in Fig. 6 and thus, we can have $g(Z)$ as $\frac{11}{8}$ with their mapping relationships as $\{(v_2,v_1),(v_2,v_4),(v_3,v_1)\}$. Iteration 4 in Table 3 shows that since the values of $\delta(v_2)$ and $\delta(v_5)$ are negative, $h(Z)=0$.

Algorithm 2. OOB: obtaining optimal backbones
Input: A reduction graph $G=(V,E)$
Output: Backbone mapping X

1: $P.NB \leftarrow V$
2: $P.B \leftarrow \phi$
3: $max \leftarrow 0$;
4: Construct a heap H according to the values of evaluation function $f(\cdot)$;
5: Insert P into H;
6: **while** heap is not empty **do**
7: Remove node P from H;
8: Branch P;
9: **for** each child Q of P **do**

10: **if** $g(Q) > max$ **then**
11: $max \leftarrow g(P)$;
12: $Best\_P \leftarrow P$;
13: remove all nodes in H which f less than max;
14: **if** $h(Q) \neq 0$ **then**
15: Insert Q into H;
16: Return $Best\_P.M$

In light of the functions derived, the OOB algorithm performs the branch-and-bound strategy by selecting the tree node with the maximum $f(\cdot)$ value for branching its child nodes, and uses the maximum $g(\cdot)$ as the bound. The maximal value of $g(\cdot)$ is updated according to the search in the state space tree. Therefore, the variable *max* records the maximum $g(\cdot)$ among current nodes so far. A node i will not be allowed to extend its child nodes for possible solutions if *f(i)* is smaller than *max* because it is impossible that the goal node is in the subtree of node i. Following the above principles, the OOB algorithm iteratively expands child nodes of those nodes whose $f(\cdot)$ is larger than *max* until no any node can be branched. If tree node P has the maximum *g(P)* and its *h(P)* is zero, this tree node is the goal node. Hence, we could have the optimal set of backbones as $P.B$. Given the backbone set $Z.B$ and the non-backbone set $Z.NB$, it is possible to derive the maximum benefit using the same procedure as the mapping phase of the BM algorithm.

Example 11. Fig. 7 shows a snapshot of deriving the optimal solution by the state space tree based on the five-query reduction graph in Fig. 5. The number associated with a tree node represents the order to which this node is expanded. In the beginning, there is only one root node, and this root node is expanded. For all nodes in level 2, since node 2 has the largest f value (i.e., 103/24) than the other nodes, node 2 are expanded. Since it has the largest f values of all nodes, node 3 is then expanded. Since the h values of all child nodes of node 3 are 0, these child nodes do not need to be expanded. Among these child nodes, node Z has the largest g value such that *max* is setting by 31/24. Since the current maximum g value is 31/24, each node whose f value is smaller than 31/24 will be pruned. Therefore, after expanding node 4, two of its children (marked with X) are pruned since their f values are smaller than $max = 31/24$.

Analysis. Since the OOB algorithm adopts the branch-and-bound strategy, the efficiency of finding optimal solution is determined by the design of the bounding function. Experimental results show that the proposed bounding function can efficiently find the optimal solution than the brute-force approach. However, in the worst case, it is still necessary to search the whole solution space to get the optimal solution. The size of the whole solution space is $\sum_{k=1}^{n} C(n,k) = 2^n - 1$, where n is the number of query trees and $C(n,k)$ denotes that k query trees are selected as the backbones. Thus, the time complexity for searching the optimal solution is $O(2^n)$.

4.3. Maintenance mechanism

This section deals with dynamic scenarios in which queries may submit or leave dynamically. The proposed algorithms could be periodically performed. However, during the time period of the execution of our proposed algorithms, users can still dynamically issue or cancel queries. Thus, this study proposes a maintenance mechanism. There are two cases to be considered: a new query being submitted and a query being canceled.

Case 1. When a new query tree arrives

When a new query tree is issued to wireless sensor networks, it is necessary to check how much benefit could be achieved by the new query tree if this new query tree was put in the backbone set. The benefit of the new query tree T_i should be determined if query tree T_i is in the backbone set. For each non-backbone tree T_j, check the corresponding mapping set M_j. If non-backbone tree T_j shares some edges with the new query tree T_i, examine these edges and derive the benefit between non-backbone T_j and backbone T_i. If the label of an edge (v_j, v_i) does not have any common sensors to the existing edges in M_j, include this edge (v_j, v_i) in M_j. However, since adjusting the routing paths of accessing intermediate results requires some message overheads, do not include this edge in M_j. Thus, the total benefit achieved by setting T_i as a backbone is formulated as $\gamma_B(v_i) = \sum_{v_j \in NB} \sum_{(v_j,v_i) \in M_j} w(v_j, v_i)$. On the other hand, it is necessary to determine how many messages will be reduced if query tree T_i is put in the non-backbone set and accesses intermediate results of backbones. This benefit, denoted as $\gamma_{NB}(v_i)$, can be derived using the same way as determining the edge weight between two query trees. Hence, $\gamma_{NB}(v_i)$ is formulated as $\sum_{v_k \in B} w(v_i, v_k)$. According to $\gamma_{NB}(v_i)$ and $\gamma_B(v_i)$, it is easy to decide whether query tree T_i should be in the backbone set or not. Explicitly, if $\gamma_{NB}(v_i) \leq \gamma_B(v_i)$, query tree T_i should become a backbone. Otherwise, T_i is a non-backbone.

Case 2. When a query tree leaves

Existing query trees may be stopped by users. In this case, it is necessary to perform the following procedure. If the left query tree is in the non-backbone set, simply let the query tree stop, as this has no effect on existing queries. On the other hand, if the query tree left is a backbone, we should compare the reduction benefit of keeping this backbone and the cost reduced by stopping

this query tree. The total benefit of keeping the query tree is estimated as $\sum_{v_j \in NB} \sum_{(v_j,v_i)\in M_j} w(v_j, v_i)$. Moreover, if the query tree T_i is terminated immediately, the number of average messages reduced per time unit is $\frac{N(T_i)}{E(T_i)}$. Consequently, the query tree T_i should be kept as a backbone even T_i is terminated, if the following condition is satisfied:

$$\frac{N(T_i)}{E(T_i)} \leq \sum_{v_j \in NB} \sum_{(v_i,v_j)\in M_j} w\left(v_i, v_j\right). \tag{6}$$

Note that if query tree T_h is stopped, a non-backbone tree must identify the subtrees that is effected by the deletion of query tree T_h. These subtrees should then stop accessing intermediate results of query tree T_h.

5. Performance evaluation

This section evaluates the performance of the proposed algorithms and compares them with other methods. All experiments were conducted using both synthesis and real datasets. Section 5.1 describes the simulation model, while Section 5.2 examines the effects of sharing intermediate results. Section 5.3 compares the performances of the BM and OOB algorithms, and Section 5.4 presents a sensitivity analysis of the BM and OOB algorithms.

5.1. Simulation model

For the synthesis dataset, a simulator for a wireless sensor network was built and implemented in Java on a Linux platform with a 2.7-GHz AMD Athlon CPU and 2 GB of RAM. In the simulation, 500 sensors were uniformly randomly deployed in a 500×500-m^2 region. The sink was located at the left-top corner of the region, and the transmission range of the sensors was 50-m, according to the specification of micaZ [7]. When a query was submitted to the sink, TAG [28] was used to form a query tree where the root node is the sink. The query range refers to those sensor nodes whose sensing data represent the data sources of one query tree. In this experiment, the query range of each query was represented as a rectangle and the locations of query ranges were randomly determined. Note that this kind of query is called a range query, which is a standard query type supported by TinyDB [38].

Since the proposed algorithms were designed using the concept of sharing the intermediate results among query trees, queries were generated in a bottom-up fashion to control the amount of intermediate results between queries. To decide a set of sensors for generating common intermediate results, we randomly selected a sharing region which is a $r \times r$ rectangle. Note that there are

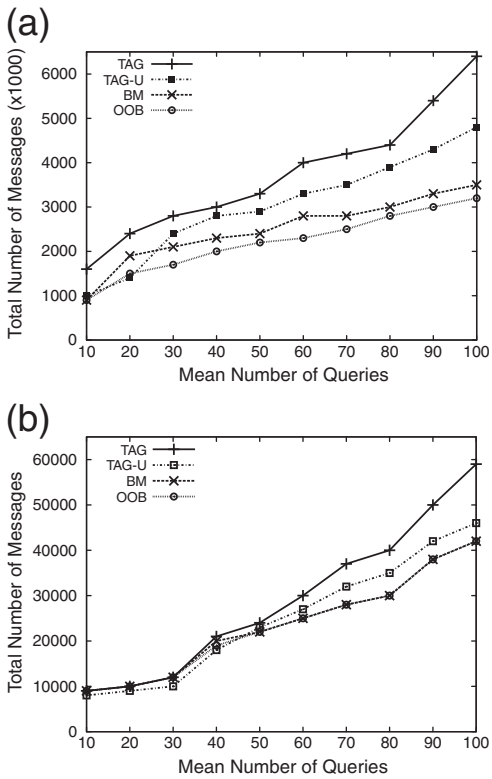

Fig. 9. Total number of messages for TAG, TAG-U, BM, and OOB. (a) The synthesis dataset and (b) the real dataset.

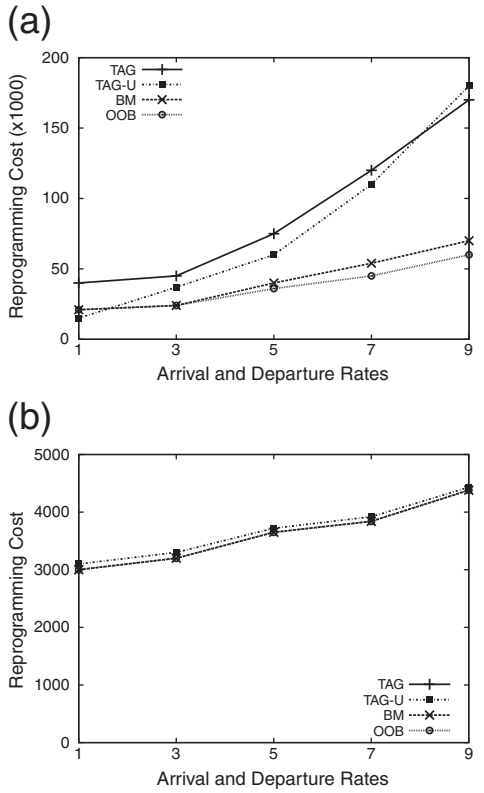

Fig. 10. Reprogramming cost of messages of TAG, TAG-U, BM, and OOB with the arrival and departure rates varied. (a) The synthesis dataset and (b) the real dataset.

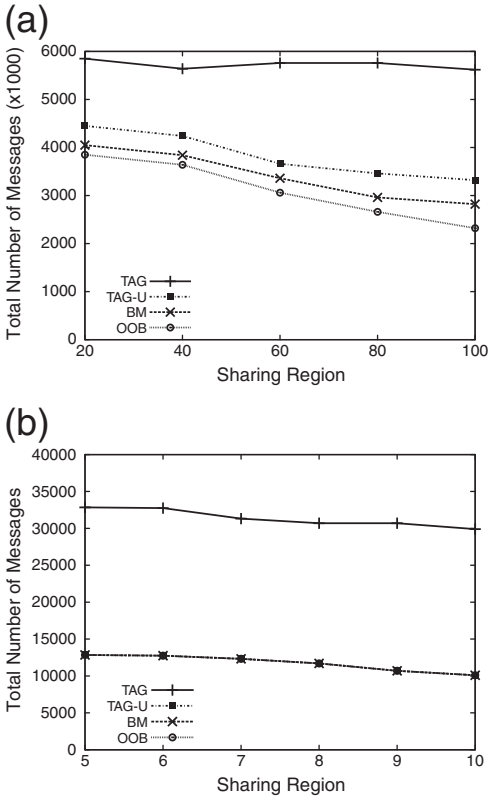

Fig. 11. Total number of messages of TAG, TAG-U, BM, and OOB with the sensing range varied. (a) The synthesis dataset and (b) the real dataset.

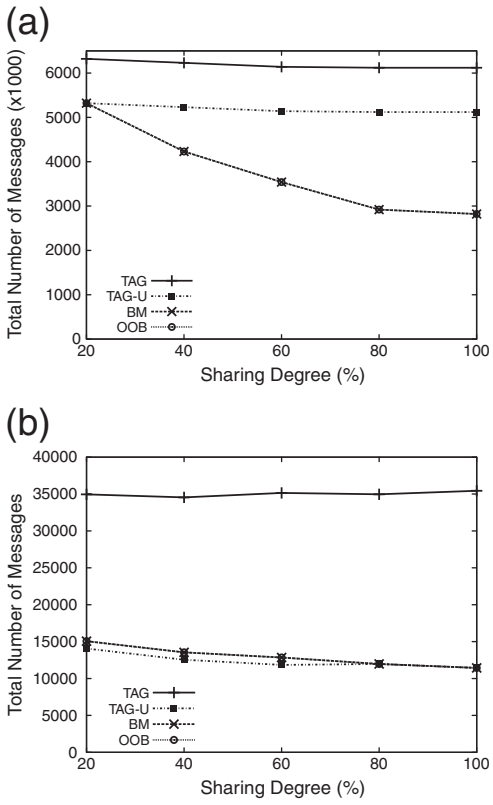

Fig. 12. Total number of messages of TAG, TAG-U, BM and OOB with the sharing degree varied. (a) The synthesis dataset and (b) the real dataset.

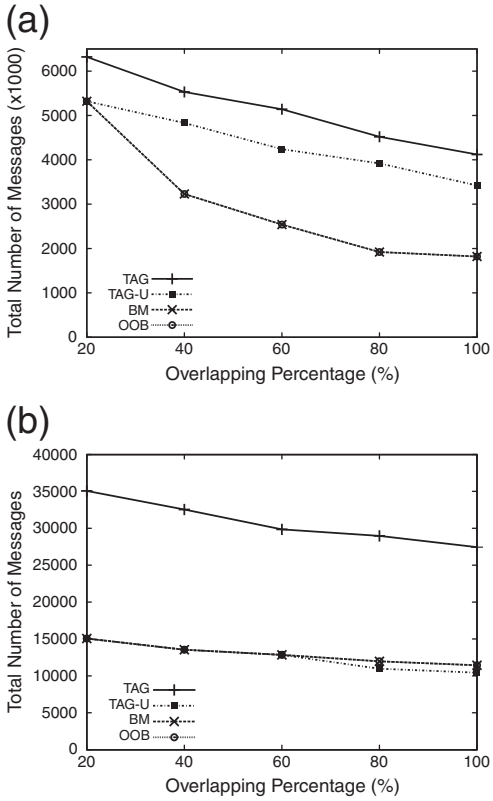

Fig. 13. Total number of messages of TAG, TAG-U, BM and OOB with the overlapping percentage varied. (a) The synthesis dataset and (b) the real dataset.

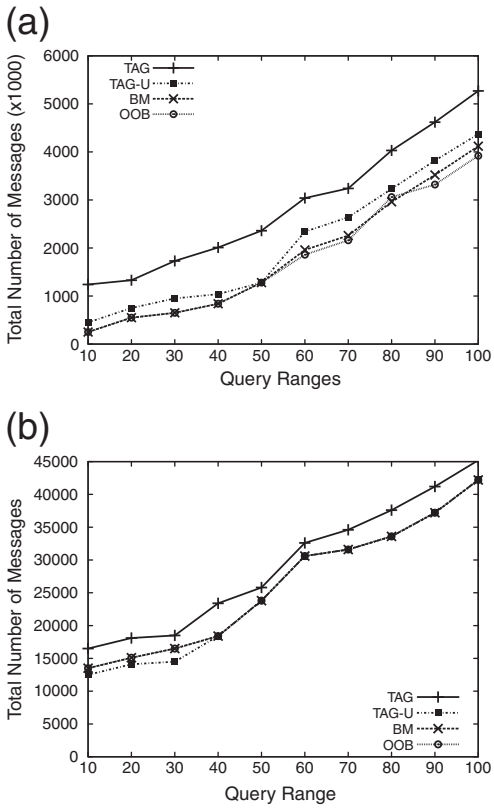

Fig. 14. Performances for TAG, TAG-U, BM and OOB with the query range varied. (a) The synthesis dataset and (b) the real dataset.

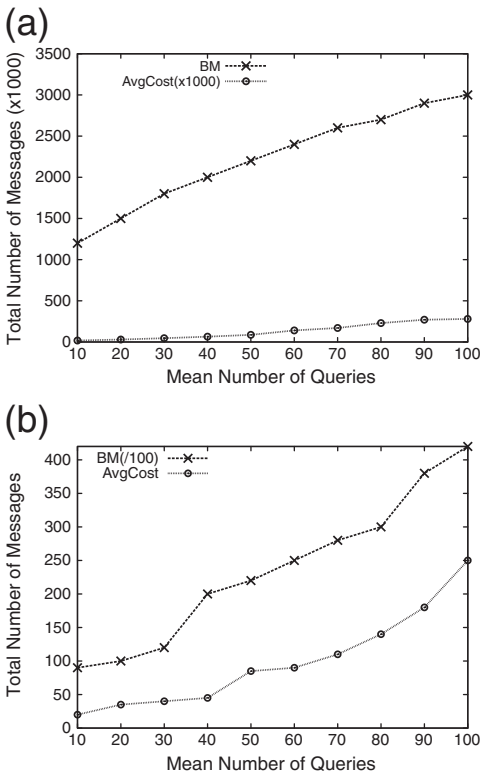

Fig. 15. Average cost and the total number of messages. (a) The synthesis dataset and (b) the real dataset.

many combinations of intermediate results of sensor nodes in a sharing region. Setting $r=3$, Fig. 8 shows a sharing region with 3×3 sensors. To generate a possible combination, a routing tree was generated for these sensor nodes in a sharing region. In this routing tree, a spanning tree was generated which leaf nodes are viewed as the source nodes and non-leaf nodes are viewed as relayed nodes. For example, bold arrows in Fig. 8 are edges of a routing tree where alphabets associated with nodes are the node identifications and the strings in nodes are the intermediate results. Note that node a, b, c, and h are leaf nodes of this routing tree so that these four nodes are viewed as the source nodes. For a routing tree, given a sharing degree k, *a valid subtree* was selected which contains $\lceil k\times r^2\rceil$ nodes of this routing tree. A valid subtree satisfies that all intermediate results in all relay nodes in this subtree should contain the source nodes in this subtree. For example, let k be 60%. The subtree contain $\lceil 60\%\times 3^2\rceil=6$ nodes. Fig. 8 shows that the subtree in the shaded region is a valid subtree with 6 nodes since all intermediate results (i.e., a, ab, and abh) are from the source nodes (i.e., a, b, and h). Note that it is possible that no valid subtree could be derived from this routing tree. In this case, we enumerate another routing tree and then find a valid subtree again. If all the routing trees cannot find a valid subtree with given requirements, another sharing region will be found and repeat the same procedure to select a valid subtree. At last, given the query range and an overlapping percentage *op*, generate n queries in which *op*-percent queries contain this valid subtree to guarantee the intermediates results that could be shared.

The simulation time was set to be 1000 units. To simulate the dynamic workloads, queries may join or leave every 50 units. The sampling period of queries were uniformly distributed in [2,14]. The arrival and departure of queries followed the Poisson process at arrival rate μ and departure rate λ. To ensure the number of query trees in the network was stable, the default value of the arrival rate and the departure rate were $\mu=3$ and $\lambda=3$, respectively. Under this setting, given the mean number of query trees, the total number of query trees in the network remained stable with respect to the mean.

The real dataset used in the following experiments is the public available Intel Lab dataset [22]. The light readings of 54 sensors are selected from this dataset for the experiments. In this dataset, due to missing readings and asymmetric communication between two sensor nodes, we fill missing readings by previous readings and consider two sensor nodes to be connected if the probability of packet loss was less than 50%. The whole map was divided into 100×100 grids. All experimental results are the average performance from readings of ten days. Note that the synthetic dataset was used to simulate a large-scale sensor networks, whereas the real dataset was used to evaluate the performance in a real world environment. The default value of the arrival rate and the departure rate were $\mu=3$ and $\lambda=3$, respectively.

For comparison purposes, *TAG* refers to the scenario in which queries were performed as usual without sharing intermediate results. On the other hand, *TAG-U* refers to deriving the union of all queries submitted at the sink first, and then injecting this query

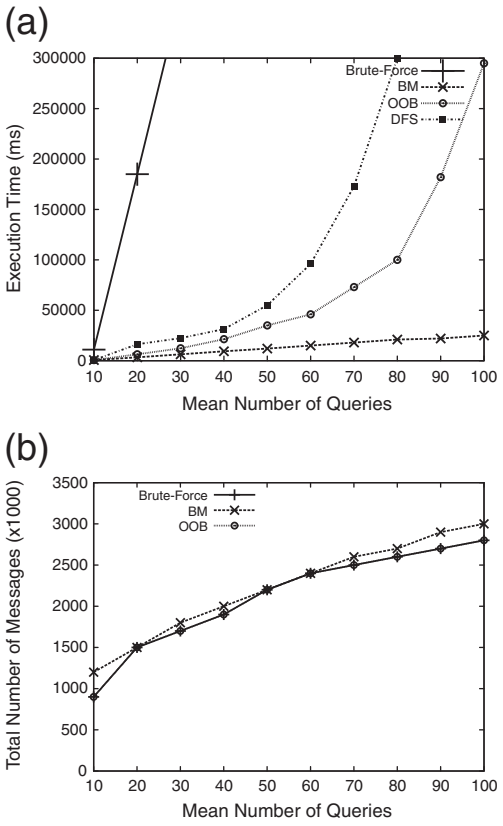

Fig. 16. Performance comparison of Brute-Force, BM and OOB with the mean number of queries varied in the synthesis dataset. (a) Execution time and (b) number of messages.

to the network via TAG. The optimal solution of backbone selection was obtained by a brute-force algorithm, denoted as *Brute-Force*. *Brute-Force* algorithm calculates all possible sharing cases and chooses the case with the minimal number of messages as the optimal solution. This study uses two performance metrics, *total number of messages* and *execution time*, to evaluate experimental results. The total number of message evaluates the number of messages involving in query processing. The execution time indicates the time to derive the sets of backbones and non-backbones.

In the following experiments, the default values of the query range were set as 100×100 m^2 and 10×10 grids, and that of the sharing regions were set as 60×60 m^2 and 6×6 grids for synthesis and real diastases, respectively. Moreover, the default value of the sharing degree is set to 60%, the default value of the overlapping percentage is set to 60%, and the default value of mean number of queries is set to 60.

5.2. The effect of sharing intermediate results

This section first evaluates the total number of messages with the mean number of queries varied. Fig. 9(a) shows that the total numbers of messages for all approaches increase with the mean number of queries. The total number of messages incurred by BM and OOB are much fewer than in TAG and TAG-U, showing the advantage of sharing intermediate results. Fig. 9(b) shows the results of the real dataset, which is similar to the one of synthesis dataset.

The section next evaluates the reprogramming cost, which represents the number of messages incurred by reconfiguring routing trees when a query tree arrives or leaves our system. Note that in the cases when the mean number of queries is fewer, the total number of messages incurred by TAG-U is slightly fewer than BM and OOB. Note that when query trees arrive or depart, TAG-U must compute the union from a set of new queries and derive the whole new query trees. Thus, the reprogramming cost of TAG-U increases as the number of query trees increases. Fig. 10 shows the reprogramming cost of BM and TAG-U with the arrival and departure rates varied. As mentioned above, the number of query trees arriving or leaving follow a Poisson process; therefore, the arrival and departure rates here represent the parameters for Poisson process. Different from the results in the synthetic datasets, reprogramming costs in the real datasets do not differ too much in any case. The reason could be the size of query trees are related smaller than a large-scale network. Thus, the reprogramming cost of all approaches is likely the same. Overall, when the arrival and departure rates are small, the reprogramming cost of TAG-U is smaller than BM. With the increasing of arrival and departure rates, the reprogramming cost of TAG-U increases significantly since it needs to modify the whole trees.

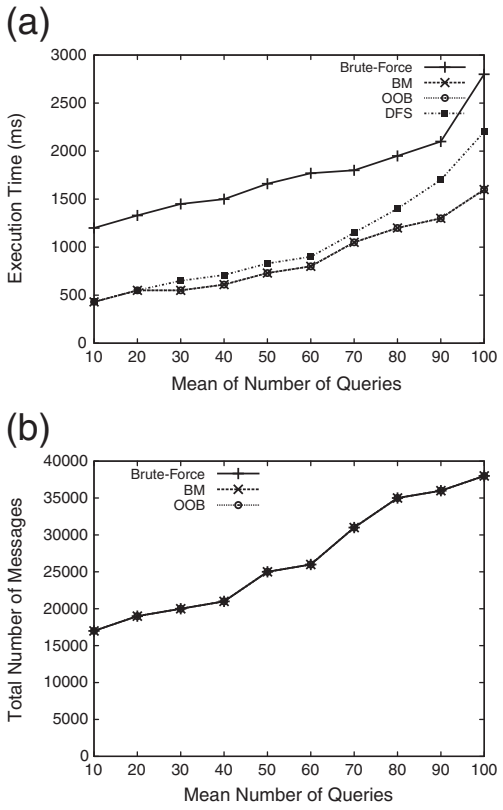

Fig. 17. Performance comparison of Brute-Force, BM, OOB, and DFS with the mean number of queries varied in the real dataset. (a) Execution time and (b) number of messages.

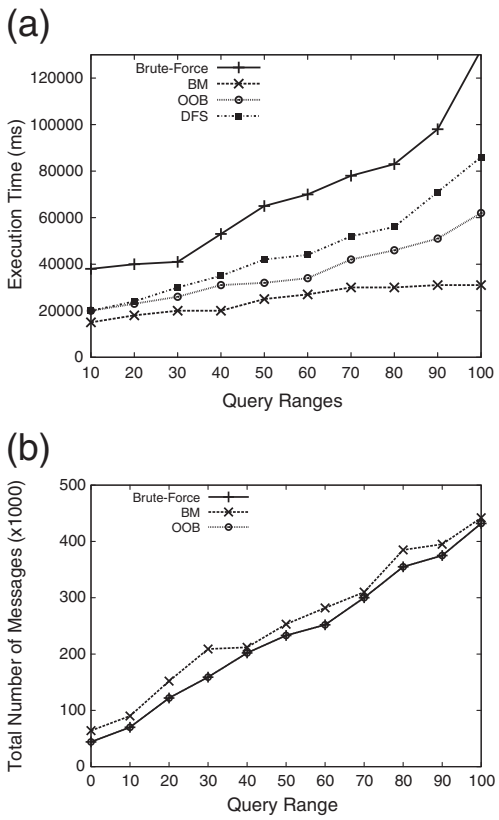

Fig. 18. Performance comparison of Brute-Force, BM, OOB, and DFS with the query range varied in the synthesis dataset. (a) Execution time and (b) total number of messages.

The following experiments examine the effects of the amount of intermediate results shared among query trees. We first investigate the effect of the sensing range. Fig. 11(a) and (b) show the experimental results for the synthesis and real dataset. The total number of messages for TAG-U, BM, and OOB decreases when the sensing ranges become larger, while that of TAG almost keeps constant. It is worth mentioning that in Fig. 11(b), the total number of messages of TAG-U is slightly larger than BM and OOB (around 500–700). In a small-scale network, since the selection of intermediate results are much fewer than a large-scale network, the topology of the tree derived by TAG-U does not differ too much than the topology derived by BM and OOB such that TAG-U could almost reduce as the same amount of messages than BM and OOB. Note that the sensing ranges represent how many sensors are used to generate intermediate results. These results indicate that BM and OOB can exploit intermediate results to reduce the total number of messages efficiently.

Next, the effects of the sharing degree are examined. Fig. 12(a) and (b) show the performance of TAG, TAG-U, BM and OOB with the sharing degree varied in the synthesis and real dataset, respectively. The total number of messages in BM and OOB decrease as the sharing degree increases, which is consistent with the fact that query trees can share more intermediate results when the sharing degree becomes larger. The effect of BM and OOB outperform TAG-U when the sharing degree is large, which the advantage of sharing intermediate results in algorithm BM and OOB.

The effect of the overlapping percentage is discussed. Fig. 13 (a) and (b) show that the total number of messages reduces when the overlapping percentage increases. A larger overlapping percentage increases the possibility of the common data sources among multiple queries. Therefore, BM and OOB are likely to share intermediate results to reduce the total number of messages.

This study also investigates the performances of TAG, TAG-U, BM and OOB for various query ranges. Fig. 14(a) shows that both BM and OOB outperform TAG and TAG-U in the synthesis dataset. Fig. 14(b) shows the similar results in the real dataset. The difference between TAG-U and the proposed methods is not prominent with the query range varied. This indicates that a larger query range leads to a larger query tree. Compared with the experimental results above, this suggests that the reduction of the total number of messages is primarily decided by the amount of the common intermediate results among query trees.

5.3. Performance comparison of algorithms BM and OOB

The results of the above experiments demonstrate that BM and OOB can significantly reduce the total number of messages. This section evaluates the efficiency of the BM and OOB algorithms in terms of execution time. Moreover, to assess the optimality of OOB, this study implements the Brute-Force algorithm to obtain the optimal set of backbone trees and the best mapping between

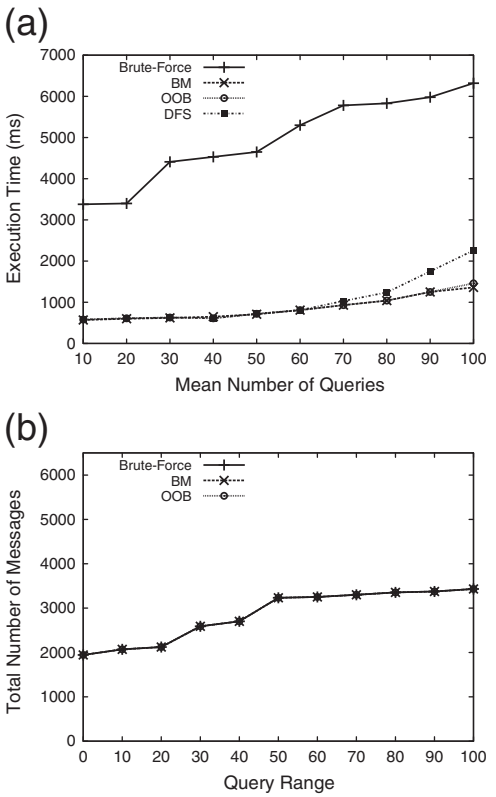

Fig. 19. Performance comparison of Brute-Force, BM, OOB, and DFS with the query range varied in the real dataset. (a) Execution time and (b) total number of messages.

non-backbone and backbone trees. We will evaluate the optimality in terms of the total number of messages. On the other hand, to evaluate the effectiveness of the bounding function of OOB, we also implement a naive bounding strategy, denoted as DFS. In DFS, nodes are expanded in depth-search-first order. As same as OOB, DFS keeps the maximum f value in *max* and never expands nodes which g values are smaller than *max*.

First, since BM and OOB exploit a cost function to estimate the total number of messages incurred, it is necessary to examine the validation of the cost function. This experiment considers the fixed set of query trees since the average cost changes when a query tree enters or leaves the system. Fig. 15 shows that the total number of messages is proportional to the average cost in all cases, which implies that the average cost reflects the total number of messages when we consider the fixed set of query trees.[2] Since the whole monitoring duration can be decomposed into small sub-durations in which the set of query trees are fixed, thus the average cost can be effectively used to reduce the total number of messages.

This study also investigates how the execution times of Brute-Force, BM, OOB, and DFS methods vary with the mean number of queries. Fig. 16(a) shows the execution times of these algorithms in the synthesis dataset, which increase exponentially with the number of queries in OOB and Brute-Force. In contrast, due to the greedy nature of BM, its execution time increases linearly, thereby demonstrating its scalability. On the other hand, it can be seen that DFS needs more time than OOB. Note that OOB uses the bounding function to lead the procedure of node expansion. Without the proposed bounding function, DFS needs longer execution time to find the optimal solution than OOB does. Fig. 16(b) shows the total number of messages for Brute-Force, BM and OOB.[3] They exhibit very similar numbers of messages while BM incurs slightly more messages than the optimal solution. This implies that BM can also provide the solution closer to the optimal one. In the real dataset, Fig. 17(a) shows that the execution time of BM, OOB, and DFS are faster than that of Brute-Force even in a small network, while Fig. 17(b) demonstrates the optimality of OOB.

We next conducted an experiment in which the query range was varied. Fig. 18(a) shows that the execution times of Brute-Force, BM, OOB, and DFS in the synthesis dataset. This figure indicates that the execution times of OOB, DFS, and Brute-Force increase much more than that of BM. Similar to the results above, Fig. 18(b) shows that the messages incurred by these four approaches are similar. This implies that our proposed approaches could achieve optimality efficiently. In the real dataset, Fig. 19 shows the execution time of Brute-Force also increases with the increasing of mean number of queries. The execution time of BM and OOB just increase slightly. However, the execution time of DFS may possibly be longer than OOB when the mean number of

[2] The values of AvgCost and BM in Fig. 15 are scaled for ease to presentation.

[3] The number of messages is decided by the members of and the mapping of non-backbones and backbones. Since DFS and OOB derive the same solution, only the results of OOB are presented here.

queries increases. This supports that the proposed bounding function works well in OOB. To sum up, in a smaller network, the BM and OOB algorithms could be adopted to reduce the total number of messages efficiently.

Fig. 20 shows the effects of overlapping. The execution times of BM, OOB, and DFS are much shorter than that of Brute-Force. This is because the number of common data resources increases as the sharing degree increases. As such, the total number of messages of these approaches decreases significantly. In addition, BM and OOB could have the optimal solution. As the same as the previous experimental results, DFS needs more time than OOB slightly. In the real dataset, Fig. 21 shows that with a larger sharing degree, a longer execution time is required to identify the sharing relation among query trees. Note that both BM and OOB can still derive optimal solutions in a more efficient manner than the Brute-Force method.

6. Discussion

This paper focuses on the problem of sharing intermediate results among multiple queries to reduce the number of messages involved. This study proposes the BM and OOB algorithms to first find non-backbones and backbones, and then derive the mapping relation. Experimental results show that BM and OOB could take advantage of sharing intermediate results to reduce the number of messages. The performance of these algorithms in different size of networks may vary. Small-scale networks caused small-sized query trees, allowing the BM and OOB algorithms to reach almost the same performance. On the other hand, query trees in large-scaled networks could have various kinds of possible combinations of backbones and non-backbones. In this case, the OOB algorithm could lead to slightly fewer number of messages (less than 10%) than the BM algorithm; however, the BM algorithm requires less execution time than the OOB algorithm. These results show the good solution quality and the good scalability of algorithm BM.

Some limitations of this paper should be discussed here. First, the methodology limitation is discussed. In backbone mapping, a non-backbone can only access intermediate results of one backbone for the same data source. A possible extension is that a non-backbone could access the same data source from more than one backbone in different time slots. This extension could further reduce the total number of messages since a non-backbone could share intermediate results from several backbones. The other extension is the design of the bounding function used in the OOB algorithm. Although experimental results show that using this bounding function can outperform a naive DFS approach, the execution time still is much longer than BM. Therefore, another extension is to derive a more sophisticated bounding function to reduce the execution time significantly. Finally, some nodes in a backbone may carry more communications that other nodes since they need to transmit their intermediate results. Currently, this

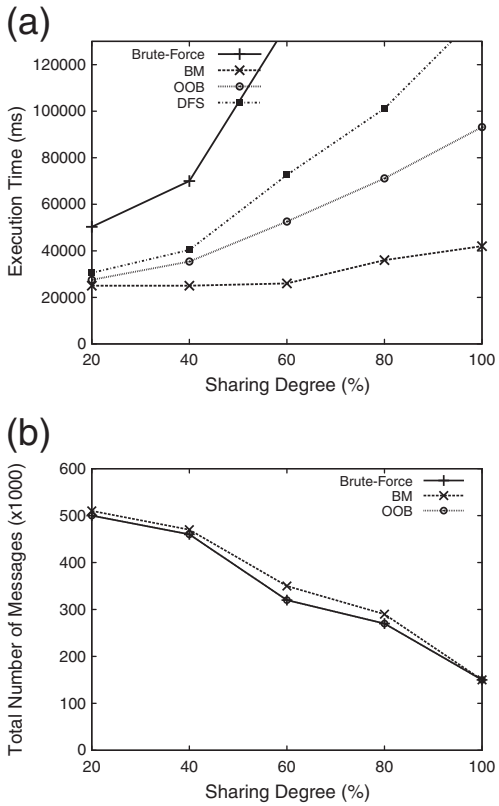

Fig. 20. Performance comparison of Brute-Force, BM, OOB, and DFS with the query range varied in the synthesis dataset. (a) Execution time and (b) total number of messages.

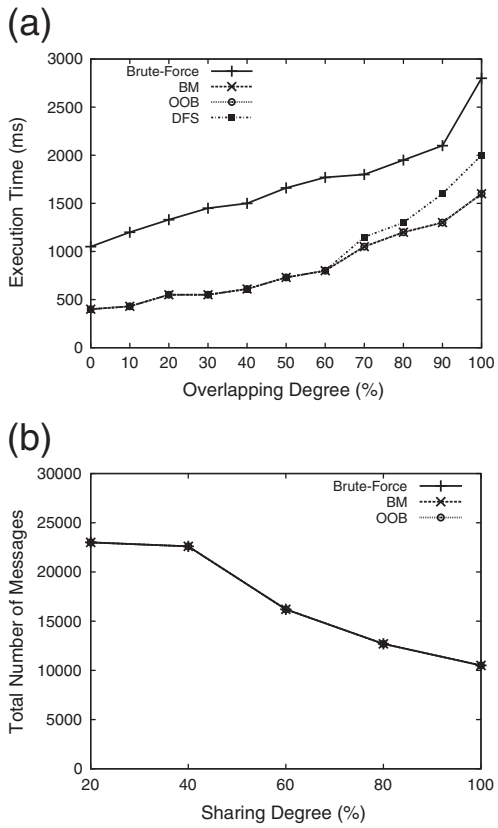

Fig. 21. Performance comparison of Brute-Force, BM, OOB, and DFS with the query range varied in the real dataset. (a) Execution time and (b) total number of messages.

problem could be solved by periodically re-submitting all queries in the proposed system. Energy-aware routing algorithms could be used to construct query trees with energy-sufficient nodes [17,44]. Using these re-constructed query trees, the BM and OOB algorithms could be executed to select new backbones to balance such overhead of energy consumption.

Second, this section also discusses the experimental limitations. The experiments in this study adopted *the total number of messages* as a metric, which is also widely used to approximate the energy consumption [8,11–13]. The MicaZ specification [7] indicates that the energy consumption of a sensor node is mainly affected by radio transmission and reception. Therefore, this metric could reflect the most dominate part of energy consumption. Moreover, when there are multiple queries in a sensor network, most network traffic consists query results to the sink. Thus, this metric could be a reasonable metric to evaluate the performance of the proposed approaches. However, to concentrate on evaluating the effects of sharing multiple queries, the proposed model ignores some minor energy consumptions activities, such as sensing, maintaining and detecting neighboring sensors, dealing with packet collision, and so on. Future research should further consider these factors by implementing the proposed approaches on networking simulators such as TAG simulator or NS-2. To test the effects of sharing intermediate results among queries, the proposed simulation model uses sharing ranges and sharing degrees to control the overlap of data sources. In reality, the number of intermediate results depends on the number of common data sources of queries. There could be other possibilities for sharing data sources. It is also possible that only few intermediate results could be shared. The experimental results in this paper may differ those in the real system. The effects of these possibilities should be further investigated by field tests.

7. Conclusions

This paper presents a method of sharing intermediate results among multiple aggregate queries to reduce the number of messages in a wireless sensor network. Given a set of query trees, sharing intermediate results among queries makes it possible to determine sets of backbones and non-backbones that minimize the number of messages incurred. This study first formulates the objective cost function for this problem. In light of the objective cost function, this study derives a reduction graph that reflects possible cases of sharing intermediate results among query trees and their associated benefits. Along with the reduction graph, this study proposes a heuristic BM algorithm. Further, the OOB algorithm uses branch-and-bound strategy to obtain the optimal solution. Comprehensive experiments were conducted on both the synthesis dataset and the real dataset. Experimental results show that by sharing intermediate results among queries, the BM and OOB algorithms are able to significantly reduce the number of messages, thereby saving a considerable amount of energy. Furthermore, the solution obtained by BM algorithm is very close to

the optimal one derived by the OOB algorithm, confirming the solution quality of the BM algorithm. In addition, with the design of the bounding function, the OOB algorithm is able to derive the optimal solution more efficiently than the brute-force approach as the number of queries increases.

Acknowledgement

Wen-Chih Peng was supported in part by the National Science Council, Project No. 97-2221-E-009-053-MY3, by Taiwan MoE ATU Program, by ITRI-JRC, Project No. 100-EC-17-A-05-01-0626, by D-Link and by Microsoft

References

[1] A. Sharaf, J. Beaver, A. Labrinidis, K. Chrysanthis, Balancing energy efficiency and quality of aggregate data in sensor networks, VLDB Journal 13 (4) (2004) 384–403.
[2] G. Amato, S. Chessa, C. Vairo, MaD-WiSe: a distributed stream management system for wireless sensor networks, Software — Practice and Experience (SPE) 40 (5) (2010) 431–451.
[3] Q. Cao, T.F. Abdelzaher, T. He, J.A. Stankovic, Towards optimal sleep scheduling in sensor networks for rare-event detection, Proceedings of International Conference on Information Processing in Sensor Networks (IPSN), 2005, pp. 20–27.
[4] Q. Cao, T.F. Abdelzaher, J.A. Stankovic, T. He, The LiteOS operating system: towards Unix-like abstractions for wireless sensor networks, Proceedings of International Conference on Information Processing in Sensor Networks (IPSN), 2008, pp. 233–244.
[5] M. Chang, P. Bonnet, Meeting ecologists' requirements with adaptive data acquisition, Proceedings of Proceedings of ACM International Conference on Embedded Networked Sensor Systems (SenSys), 2010, pp. 141–154.
[6] D. Chu, A. Deshpande, J.M. Hellerstein, W. Hong, Approximate data collection in sensor networks using probabilistic models, Proceedings of IEEE International Conference on Data Engineering (ICDE), 2006, pp. 48–59.
[7] Crossbow Technology Co. MicaZ Datasheet. http://www.xbow.com, 2009.
[8] T. Dang, N. Bulusu, W. chi Feng, S. Park, DHV: a code consistency maintenance protocol for multi-hop wireless sensor networks, Proceedings of European Conference on Wireless Sensor Networks (EWSN), 2009, pp. 327–342.
[9] A. Deshpande, S. Madden, MauveDB: supporting model-based user views in database systems, Proceedings of ACM SIGMOD, 2006, pp. 73–84.
[10] C. Fragouli, J. Widmer, J.-Y.L. Boudec, A network coding approach to energy efficient broadcasting: from theory to practice, Proceedings of IEEE International Conference on Computer Communications (INFOCOM), 2006, pp. 1–11.
[11] T.M. Gil, S. Madden, Scoop: an adaptive indexing scheme for stored data in sensor networks, Proceedings of International Conference on Data Engineering (ICDE), 2007, pp. 1345–1349.
[12] H. Gupta, V. Navda, S. Das, V. Chowdhary, Efficient gathering of correlated data in sensor networks, ACM Transaction on Sensor Network 4 (1) (2008) 1–31.
[13] H. Haanpää, A. Schumacher, T. Thaler, P. Orponen, Distributed computation of maximum lifetime spanning subgraphs in sensor networks, Proceedings of International Conference on Mobile Ad-hoc and Sensor Networks (MS), 2007, pp. 445–456.
[14] C.-C. Hung, W.-C. Peng, Clustering object moving patterns for prediction-based object tracking sensor networks, Proc. of ACM Conference on Information and Knowledge Management (CIKM), 2009, pp. 1633–1636.
[15] C.-C. Hung, W.-C. Peng, Y.S.-H. Tsai, W.-C. Lee, Exploiting spatial and data correlations for approximate data collection in wireless sensor networks, ACM International Workshop on Knowledge Discovery from Sensor Data (SensorKDD), 2008, pp. 1–8.
[16] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE Communication Magazine 40 (8) (2002) 102–114.
[17] K.Y. Jang, K.T. Kim, H.Y. Youn, An energy efficient routing scheme for wireless sensor networks, Proceedings of IEEE International Conference on Computational Science and Applications (ICCSA) 0 (2007) 399–404.
[18] K.C.K. Lee, W.-C. Lee, B. Zheng, J. Winter, Processing multiple aggregation queries in geo-sensor networks, Proceedings of International Conference on Database Systems for Advanced Applications (DASFAA), 2006, pp. 20–34.
[19] C. Liu, K. Wu, J. Pei, A dynamic clustering and scheduling approach to energy saving in data collection from wireless sensor networks, Proceedings of IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2005, pp. 374–385.
[20] S. Liu, K.-W. Fan, P. Sinha, CMAC: an energy efficient MAC layer protocol using convergent packet forwarding for wireless sensor networks, Proceedings of IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON), 2007, pp. 11–20.
[21] G. Lu, N. Sadagopan, B. Krishnamachari, A. Goel, Delay efficient sleep scheduling in wireless sensor networks, Proceedings of IEEE International Conference on Computer Communications (INFOCOM), 2005, pp. 2470–2481.
[22] S. Madden, Intel Lab Data, http://berkeley.intel-research.net/labdata. 2004.
[23] C. Mala, S. Selvakumar, Construction of an optimal multicast tree for group communication in a cellular network using genetic algorithm, Computer Communications 29 (16) (2006) 3306–3312.
[24] N.J. Nilsson, Principles of Artificial Intelligence, Morgan Kaufmann Publishers Inc, 1980.
[25] M. Rosenmüller, S. Apel, T. Leich, G. Saake, Tailor-made data management for embedded systems: a case study on Berkeley DB, Data and Knowledge Engineering 68 (12) (2009) 1493–1512.
[26] A. Rowe, K. Lakshmanan, R. Rajkumar, SAMPL: a simple aggregation and message passing layer for sensor networks, Proceedings of International ICST Conference on Wireless Internet (WICON), 2008.
[27] P.M. Ruiz, A.F. Gómez-Skarmeta, Approximating optimal multicast trees in wireless multihop networks, Proceedings of IEEE Symposium on Computers and Communications (ISCC), 2005, pp. 686–691.
[28] S. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, TAG: a tiny aggregation service for ad-hoc sensor networks, Proceedings of Symposium on Operating Systems Design and Implementation, 2002.
[29] S. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, TinyDB: an acquisitional query processing system for sensor networks, ACM Transactions on Database System 30 (1) (2005) 122–173.
[30] S. Xiang, H.-B. Lim, K.-L. Tan, Y. Zhou, Two-tier multiple query optimization for sensor networks, Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS), 2007, pp. 39–48.
[31] P.-R. Sheu, S.-T. Chen, A fast and efficient heuristic algorithm for the delay- and delay variation-bounded multicast tree problem, Computer Communications 25 (8) (2002) 825–833.
[32] N. Shrivastava, C. Buragohain, D. Agrawal, S. Suri, Medians and beyond: new aggregation techniques for sensor networks, Proceedings of ACM International Conference on Embedded Networked Sensor Systems (SenSys), 2004, pp. 239–249.
[33] T.-Y. Fu, W.-C. Peng, W.-C. Lee, Optimizing parallel itineraries for KNN query processing in wireless sensor networks, Proceedings of ACM Conference on Information and Knowledge Management (CIKM), 2007, pp. 391–400.
[34] L.A. Tang, X. Yu, S. Kim, J. Han, C.-C. Hung, W.-C. Peng, Tru-alarm: trustworthiness analysis of sensor networks in cyber-physical systems, Proc. of IEEE International Conference on Data Mining (ICDM), 2010, pp. 1079–1084.
[35] X. Tang, J. Xu, Optimizing lifetime for continuous data aggregation with precision guarantees in wireless sensor networks, IEEE/ACM Transaction on Networking 14 (4) (2008) 904–917.

[36] N. Trigoni, Y. Yao, A.J. Demers, J. Gehrke, R. Rajaraman, Multi-query optimization for sensor networks, Proceedings of International Conference on Distributed Computing in Sensor Systems (DCOSS), 2005, pp. 307–321.
[37] N. Tsiftes, J. Eriksson, A. Dunkels, Low-power wireless IPv6 routing with ContikiRPL, Proceedings of International Conference on Information Processing in Sensor Networks (IPSN), 2010, pp. 406–407.
[38] A. Woo, S. Madden, R. Govindan, Networking support for query processing in sensor networks, Communication of the ACM 47 (6) (2004) 47–52.
[39] S.-H. Wu, K.-T. Chuang, C.-M. Chen, M.-S. Chen, Toward the optimal itinerary-based KNN query processing in mobile sensor networks, IEEE Transactions on Knowledge and Data Engineering 20 (12) (2008) 1655–1668.
[40] X. Tang, J. Xu, Extending network lifetime for precision-constrained data aggregation in wireless sensor networks, Proceedings of IEEE International Conference on Computer Communications (INFOCOM), 2006, pp. 1–12.
[41] X. Tang, J. Xu, W.-C. Lee, EASE: an energy-efficient in-network storage scheme for object tracking in sensor networks, Proceedings of IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON), 2005, pp. 396–405.
[42] X.-Y. Xiao, W.-C. Peng, C.-C. Hung, W.-C. Lee, Using sensor ranks for in-network detection of faulty readings in wireless sensor networks, ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE), 2007, pp. 1–8.
[43] Y. Yao, J. Gehrke, The cougar approach to in-network query processing in sensor networks, ACM SIGMOD Record 31 (3) (2004) 9–18.
[44] M. Zimmerling, W. Dargie, J.M. Reason, Energy-efficient routing in linear wireless sensor networks, Proceedings of IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS) 0 (2007) 1–3.

Chih-Chieh Hung received the MS degree from National Chiao Tung University (NCTU), Taiwan in 2005 and the Ph.D degree in the Department of Computer Science from National Chiao Tung University, Taiwan, in 2011. During his Ph.D program, he was mainly involved in the projects related to trajectory pattern mining and applications, data management on sensor networks, and sensor-enabled cloud computing. Currently, he has already published 9 international conference papers and 2 international journal papers; especially, he received the Best Paper Award in ACM Workshop on Location-Based Social Network in 2009.
Dr. Hung serves as a PC member in IADIS Multi Conference on Computer Science and Information Systems 2011 (INFORMATICS-I2011). His research interests include trajectory pattern mining, location-based social network, sensor data management, and data mining.

Wen-Chih Peng was born in Hsinchu, Taiwan, R.O.C in 1973. He received the BS and MS degrees from the National Chiao Tung University, Taiwan, in 1995 and 1997, respectively, and the Ph.D. degree in Electrical Engineering from the National Taiwan University, Taiwan, R.O.C in 2001. Currently, he is an associate professor at the department of Computer Science, National Chiao Tung University, Taiwan. Prior to joining the department of Computer Science and Information Engineering, National Chiao Tung University, he was mainly involved in the projects related to mobile computing, data broadcasting and network data management. Dr. Peng serves as PC members in several prestigious conferences, such as IEEE International Conference on Data Engineering (ICDE), Pacific Asia Knowledge Discovering and Mining (PAKDD) and Mobile Data Management (MDM). His research interests include mobile computing, network data management and data mining. He is a member of IEEE.