Optimized Storage Placement over Large Scale Sensor Networks

Lei Xie, Sanglu Lu, Yingchun Cao, and Daoxu Chen
State Key Laboratory of Novel Software Technology
Nanjing University, Nanjing, China
lxie@nju.edu.cn, sanglu@nju.edu.cn, yccao@nju.edu.cn, cdx@nju.edu.cn

ABSTRACT
Data storage has become an important issue for energy efficient data management over sensor networks. In this paper, we investigate into the optimized storage placement problem over large scale sensor networks, aiming to achieve minimized energy cost. In order to efficiently deal with the large scale deployment area with irregular shape, we propose to utilize the “hop” as the computation unit instead of the “node”, such that the computation complexity can be greatly reduced. We propose methodologies to solve the optimization problem respectively in situations for unlimited number of storages and limited number of storages. The ultimate goal of this paper is to give a fundamental guidance of optimized storage placement for large scale sensor networks. Simulation results confirm the performance of our methodologies.

Keywords
Data Storage, Optimization, Storage Placement, Sensor Network, Large Scale

1. INTRODUCTION
Sensor networks in pervasive computing applications, such as environment monitoring, health caring and target tracking, generate a large amount of data. Generally these data are collected from sensor nodes over the sensor network and the end users retrieve them through diffusing specific queries from the sink into the network [1]. Conventionally both raw sensor data and queries are continuously generated over the sensor network. Due to the limited battery power in these sensor nodes, it will greatly increase the overall energy cost by simply forwarding all raw sensor data to the sink, moreover, this will make sensor nodes around the sink heavily used and quickly exhausted in energy. We note that currently some specially designed sensor nodes are equipped with larger storage capability than normal sensors, thus they can store a certain amount of raw sensor data in their local storages. Hence when queries are diffused into these nodes, they can process these queries, filter locally stored raw sensor data, and send out query replies to users[2]. In this way, a large amount of raw sensor data can be avoided for transmission such that the overall energy cost for data forwarding is greatly reduced. Based on the above understanding, various data-centric storage schemes [3] have been proposed to sufficiently leverage the sensors’ local storage capability to achieve energy efficiency.

However, on the other side, it has also been demonstrated not energy-efficient to locally store the collected raw sensor data in all sensor nodes. This will introduce a large amount of query diffusion cost, since the queries should be broadcasted from the sink to each of the storage nodes. We note that by appropriately deploying storage nodes over sensor networks, the heavy load of query diffusion and raw sensor data forwarding can be alleviated, through making appropriate tradeoffs between the above “pure push” and “pure pull” schemes[3]. Therefore, a “push and pull” based storage scheme is essential to extract information from sensor network, while achieving the overall energy efficiency [4]. In this scheme, some intermediate “storage nodes” are deployed over the sensor network, other ordinary nodes called “forwarding nodes” just send their raw data upward to these storage nodes along the routing tree, and queries are diffused to these storage nodes to fetch those filtered sensor data as query replies. Based on the above scenario, a challenging problem appears, that is how to place the storage nodes over the sensor network such that the overall energy efficiency can be achieved.

To deal with this problem, Sheng et al. have proposed optimized algorithms based on dynamic programming to solve the storage placement problem in the tree based model [5]. However, in some industrial or research applications where sensor networks are widely and densely deployed in a large scale approach, it is laborious and time-consuming to compute the optimized storage placement one by one based on the above algorithm. As a matter of fact, suppose the sensor nodes are uniformly distributed, it is not essential to compute the exact optimized storage placement due to the large scale deployment. Therefore, novel approaches should be proposed to simplify the computation of storage placement over the large scale sensor network, which is in irregular shape for general cases. In this paper, we investigate into this problem and propose optimized storage placement in the situations respectively with unlimited number of storages and limited number of storages. Due to the large scale
deployment, we divide the deployment area into separate partitions and leverage the “hop” as the computation unit instead of the “node”, which greatly reduces the computing complexity. By utilizing the characteristics of large scale sensor network, our solution greatly simplifies the computation complexities.

We aim to minimize the overall energy cost in the data accumulation and query by appropriately placing storage nodes over the sensor network. With unlimited number of storages, we analyze the optimized storage structure over the deployment area, and further provide the optimized parameters for storage deployment according to the irregular shapes of the deployment area. With limited number of storages, we reduce the problem into an unbounded knapsack problem. Furthermore, we respectively propose dynamic programming based algorithm and greedy approximation algorithm to effectively solve this problem.

The rest of this paper is organized as follows, Section II reviews the related work. Section III defines problems for storage node placement in the aforementioned scenario. Section IV and Section V respectively present optimized storage placement methodologies for unlimited number of storages and limited number of storages. Section VI conducts performance analysis. Section VII presents simulation results. Section VIII concludes the paper.

2. RELATED WORK

There have been a lot of research works for data-centric storage schemes over sensor network, which mainly include structured storage scheme and unstructured storage scheme. Structured storage scheme mainly utilizes a mapping function such as geographic hash function to map event information to specified geographic points, and leverages routing algorithms like GPSR [6] to route messages to corresponding storage nodes, research works including GHT [6], DIMENSIONS [7] and DIM [8] belong to this scheme. Unstructured storage scheme mainly utilizes the “push-pull” scheme to disseminate and gather information, research works including Double Ruling [9], Comb-Needles [10] belong to this scheme. Fang et al. [11] propose a hybrid storage scheme to implement the event storage and query. Trigoni et al. [12] presents a hybrid push and pull scheme for tree based structure where queries are injected from the sink. Ahn et al. [13] conduct fundamental analysis over the scalability of these two kinds of data-centric storage schemes.

Recent research works focus on the storage placement problem to enhance energy efficiency and load balance. Zhang et al. [14] propose a ring based index structure to help reduce energy consumption of storage schemes. KDDCS [15] presents a load-balanced storage scheme for sensor networks based on K-D tree. Sheng et al. present optimized algorithms based on dynamic programming to solve the storage placement problem over fixed tree model [5][16], moreover, they propose an approximation algorithm to solve the storage placement problem over randomized dynamic tree model [17]. Their dynamic programming methodology must save a large amount of intermediate state parameters, and compute from bottom to top along the tree based routing structure. While achieving a fair degree of accuracy, their optimized solutions, however, is not suitable for the storage planning over large scale sensor networks, due to the huge computation complexity for large scale deployment.

3. PROBLEM FORMULATION

Without loss of generality, we consider a data collecting scenario in a large-scale sensor network as follows: all sensor nodes are densely and uniformly distributed over a deployment area, while the sink is set at some specified position within the region in advance. The deployment area is required to satisfy the following convex property: for any deployed sensor node, each point on the straight line segment that joins the sensor node with the sink also lies within the deployment area. The exact shape of the deployment area is not restricted, either regular or irregular shape is included. We classify these sensor nodes into forwarding nodes and storage nodes, where storage nodes store sensor data and forwarding nodes simply forward their data to those storage nodes. For ease of presentation, we can also regard the sink node as a storage node. Raw sensor data collected from forwarding nodes are continuously sent upward along the routing tree to the first parent storage node they meet, and queries are diffused from the sink to all storage nodes to fetch the most recent sensed data, query replies will be sent back from storage nodes to the sink along the routing tree structure. Fig. 1 depicts the data access model with storage nodes. Due to the dense and uniform distribution of sensor nodes, for ease of analysis, we assume to deal with a uniform topology structure for the large-scale sensor network, hence the network is supposed to have a near-symmetric routing tree topology over the deployment area, which implies nodes at the same hop from the sink have similar subtree topologies.

![Figure 1: Data access model with storage nodes, the black nodes denote the storage nodes while the white nodes denote the forwarding nodes.](image)

We define the following parameters for the above scenario: for data generation, we define $r_d$ as data generate rate and $s_d$ as the raw data size; for query diffusion, we use $r_q$ to denote query rate and $s_q$ to denote query size; for query reply, we use $\alpha$ to denote reply reduction rate, for input $x$, which is the size of raw data generated by a set of nodes, function $f(x) = \alpha x$ for $\alpha \in (0, 1]$ returns the size of processed data as query replies.

For the large scale deployed sensor network, the objective is to place storage sensor nodes (and hence forwarding sensor nodes) over the deployment area with irregular shape such that the overall energy cost is minimized. Then the problems to solve are as follows: (1) Given unlimited storage.
nodes, how to deploy storage nodes so as to attain the minimum overall energy cost? (2) Given limited storage nodes (number of storage nodes = \( K \)), how to deploy storage nodes so as to attain the minimum overall energy cost? Most important of all, the corresponding solutions are required to be scalable to the large scale deployment in computation complexity, such that the optimized storage placement can be figured out in an efficient approach.

4. UNLIMITED NUMBER OF STORAGES

When the storage capability over each sensor node is large enough to hold some raw sensor data for further usage, then any sensor node has the opportunity to become a storage node, thus we can suppose to have unlimited number of storages, every sensor node over the routing tree structure can switch to the role of storage node if necessary.

4.1 Optimized Storage Placement Structure

Before analyzing on the optimized storage placement, we give the definition of unit zone. A unit zone is a fan-shaped region rooted from the sink within the deployment area, it has a complete tree based topology with the minimum granularity, which we called unit topology. Fig.2 illustrates an example of the unit zones within the gray deployment area. Due to the irregular shape of the deployment area, the maximum hop varies among the unit zones, but the degree \( \theta \) is equivalent for all unit zones. Therefore, the overall deployment area can be divided into a certain number of unit zones and the whole topology is composed of these corresponding unit topologies. The optimal storage placement inside each unit zone is independent with each other among the unit zones.

![Figure 2: Unit zone](image)

We rely on the following conclusion to obtain the optimized storage structure, which is first proposed by [5]: In the optimal tree, if \( s_i \) is a forwarding node, all of its descendants are forwarding nodes as well. If \( s_i \) is a storage node, all its ancestors are storage nodes as well. According to this conclusion, we have Theorem 1.

**Theorem 1.** With unlimited number of storages, the optimized placement structure for storage nodes in the unit zone is a fan-shaped area, where sensors inside the fan-shaped area are storage nodes, and others are forwarding nodes.

**Proof.** According to conclusion from [5], all storage nodes have storage ancestors over the tree in the optimal solution, thus we can deduce the optimized storage placement structure is a storage area surrounding the sink node. As the network topology in the unit zone is uniform and symmetric, we can prove the storage area is a fan-shaped area by contradiction. As Fig.3 shows, suppose the storage area in a unit zone is in an irregular shape instead of a regular fan shape, as the gray area depicts. Inside the storage area, we can find a fan-shaped region \( R \) with the maximum radius, then we randomly choose a storage node outside \( R \), we denote it as node \( A \). Due to the symmetry of the topology, any node with the same hop of node \( A \), e.g., node \( B \), must have the same opportunity to be storage nodes in the optimal solution. However, node \( B \) is set to be a forwarding node in the former hypothesis. Therefore we get a contradiction against the former hypothesis, hence the theorem gets proved.

![Figure 3: Proof of Theorem 1, we prove it by contradiction.](image)

4.2 Calculating Optimized Parameters

According to the conclusion from [5]: If \( \alpha r_s \geq r_s \), then the optimal tree with the minimum energy cost contains only forwarding nodes (except for the root). If \( \alpha r_q < r_s \), there exits a storage area for the optimal tree., we know that if \( \alpha r_q \geq r_s \), the optimized solution includes no storage nodes. Therefore in the remaining of this paper we only consider the \( \alpha r_q < r_s \) situation. According to Theorem 1, it is proved that the optimized storage structure inside each unit zone is a fan-shaped area around the sink, there must exist an optimized hop-count \( k_{opt} \) for the fan-shaped area, where sensors within the \( k_{opt} \) hops are storage nodes, and sensors outside the \( k_{opt} \) hops are forwarding nodes. Hence \( k_{opt} \) actually denotes the boundary of the storage area. Next we will calculate the optimized parameter \( k_{opt} \) for the optimized storage area. Without loss of generality, in the rest of this paper we utilize the equal sign “=” to denote the expected value of specified variables.

Assume the degree of a unit zone is \( \theta (0 < \theta < \pi) \), the maximum hop of a specified unit topology is \( n \), the average hop distance is \( d \), and the network node density is \( p \), thus the number of sensor nodes for the \( k \)th hop is \( N_k = \theta(2i-1)d^2p \). Now we analyze the energy consumption model. To transmit \( s \) data units, the energy costs of the sender and receiver are
\( e_{tr} \cdot s \) and \( e_{re} \cdot s \), where \( e_{tr} \) and \( e_{re} \) are the energy costs for transmitting and receiving a unit data respectively, and \( e_{tr} \) is also relevant to the distance between the sender and receiver. For simplicity of presentation, the receiving energy cost is assigned to the transmitter without changing the total energy cost. Here we use \( S_i \) to denote an arbitrary sensor node in the \( i \)th hop, when sensor \( S_i \) in the \( i \)th hop sends \( s \) data units to \( S_j \) in the \( j \)th hop, the energy cost of \( S_j \) is 0, and the energy consumed by \( S_i \) is:

\[
\begin{cases} 
(e_{tr} + e_{re}) \cdot s & \text{if } S_j \text{ is parent of } S_i \\
(e_{tr} + e_{re} \cdot c_i) \cdot s & \text{if } S_j \text{ is one of } S_i \text{'s children}
\end{cases}
\]

(1)

In the above equation \( c_i \) is the number of \( S_i \)'s children. For the following discussion, we normalize the energy costs by \( (e_{tr} + e_{re}) \) for ease of presentation. Thus, to transfer \( s \) data units from \( S_i \) to its parent, the transmitting energy will be \( s \) and to broadcast \( s \) data units to its children, sensor \( S_i \) will consume \( b_i \cdot s \) energy, where \( b_i = \frac{N_i}{e_{re} + e_{tr}} \). Based on the assumption of near-symmetric topology we can denote \( c_i \) as the average number of children for sensor nodes in the \( i \)th hop, where \( c_i = \frac{N_i}{e_{re} + e_{tr}} = \frac{1}{2} \). And we set \( |T_i| \) as the average number of sensor nodes in the subtree for any node in the \( i \)th hop (including the \( i \)th hop node itself), thus we have:

\[ |T_i| = 1 + \sum_{j=1}^{n} N_j = 1 + \frac{n^2 - 1}{2i - 1}. \]

As we know the energy cost \( E_i \) for a sensor node in the \( i \)th hop consists of three types of energy costs:

\[ E_i = E_q + E_r + E_d. \]

In the above equation, \( E_q \) denotes energy cost for query diffusion, \( E_r \) denotes energy cost for query reply, and \( E_d \) denotes energy cost for raw data forwarding. Hence based on the above energy model, we calculate the energy consumption for each \( i \)th hop, here we set the sink at the 0th hop.

We use \( k \) to denote the last hop of the storage node area. For storage nodes within the \( k \)th hop \((1 < i < k)\), we have:

\[
\begin{cases} 
E_r = r_q \alpha |T_i| s_d \\
E_d = 0 \\
E_q = b_i r_q s_g
\end{cases}
\]

(2)

Here we set the reply data size as \( \alpha |T_i| s_d \) for the snapshot query only fetches the most recent sensing data, and we set \( E_d = 0 \) because although the storage nodes generate raw data locally, they don't need to forward these data until they're sent out as query replies. For storage nodes right in the \( k \)th hop, \((i = k)\), we have:

\[
\begin{cases} 
E_r = r_q \alpha |T_k| s_d \\
E_d = 0 \\
E_q = 0
\end{cases}
\]

(3)

Here we set \( E_q = 0 \) because the last hop of storage nodes need not to diffuse queries to their forwarding node children. For forwarding nodes outside the \( k \)th hop \((k + 1 \leq i \leq n)\):

\[
\begin{cases} 
E_r = 0 \\
E_d = |T_i| s_d s_d \\
E_q = 0
\end{cases}
\]

(4)

Therefore the overall energy consumption can be depicted as:

\[
E_{total}(k) = \sum_{i=1}^{k-1} [(E_r + E_q) \cdot N_i] + E_r \cdot N_k + \sum_{i=k+1}^{n} (E_d \cdot N_i).
\]

(5)

And we let

\[
\begin{align*}
R_r &= r_q \alpha s_d \\
R_d &= r_d s_d \\
R_q &= r_q s_q
\end{align*}
\]

To calculate the optimal hop count \( k_{opt} \) for minimized overall energy consumption, we obtain

\[ \frac{\partial E_{total}(k)}{\partial k} = 0 \Rightarrow (R_d - R_r) \cdot k^2 + (R_c - R_d + 2R_q) \cdot k + C = 0, \]

where

\[ C = \frac{1 - n^2}{6} (R_d - R_r) + \frac{-2e_{tr}}{e_{re} + e_{tr}} R_q. \]

Thus \( k_{opt} \) has two possible solutions

\[ k_1 = \frac{(R_d - R_r - 2R_q) + \delta}{2(R_d - R_r)}, \quad k_2 = \frac{(R_d - R_r - 2R_q) - \delta}{2(R_d - R_r)}, \]

where

\[ \delta = \sqrt{(R_c - R_d + 2R_q)^2 - 4C(R_d - R_r)}. \]

As we deduce from \( \alpha r_q < r_d \) there exists \( R_d > R_r \), therefore we get:

\[ k_2 = \frac{1}{2} - \frac{R_d}{R_d - R_r} - \frac{\delta}{2(R_d - R_r)} < \frac{1}{2}. \]

As we have \( k_{opt} \geq 1 \) we obsolete \( k_2 \) and finally obtain the optimal hop count:

\[ k_{opt} = \frac{(R_d - R_r - 2R_q) + \delta}{2(R_d - R_r)}. \]

(6)

According to Eq. (6), it is known that the optimal hop count for the storage area inside the unit zone, i.e. \( k_{opt} \), depends on the maximum hop count of the corresponding unit topology, i.e. \( n \). Furthermore, it can be inferred that the larger value of \( n \) is, the larger value of \( k_{opt} \) can be obtained. Based on the above analysis, if there are unlimited number of storages, we can compute the optimized storage placement as follows: For a specified deployment area with irregular shape, it can be approximately divided into a certain number of fan-shaped partitions according to the irregular shape. Each fan-shaped partition is rooted from the sink and must contain no less than one unit zones. Moreover, the fan-shaped partition is relatively regular in shape as compared to the overall deployment area, thus we can average the the hop counts of leaf nodes inside the fan-shaped partition and approximate the maximum hop count as \( \hat{n} \). Therefore, we can approximately compute the optimal hop count \( k_{opt} \) for the storage area respectively for each fan-shaped partition. Then, by putting the pieces together, the whole storage area can be obtained. Fig.4 illustrates the basic idea of the method. Theoretically, when the deployment area is divided into more refined partitions, the solution is closer to the ideal optimal storage structure. In extreme case, when each of the fan-shaped partitions is exactly a unit zone, the optimal storage structure can be obtained.
5. LIMITED NUMBER OF STORAGES

In some situations we observe that not all sensor nodes have enough local storages because more storage will come along with increased hardware cost, we are only allowed to deploy a limited number of specially designed storage nodes for energy efficiency. When dealing with the storage placement problem with limited number of storages, it means that the number of storages, \( K \), is smaller than the optimal number of storages, \( K^* \), for unlimited storages, otherwise it can be reduced to the unlimited storage problem. As described above we only consider the \( or_q < r_d \) situation.

5.1 Optimized Storage Placement Structure

We first give the definition of energy reduction. The energy reduction of a node \( S_i \) means the changed energy cost for \( S_i \) to switch from a forwarding node to a storage node, while the roles of other nodes remain unchanged, we denote it as \( \Delta E(S_i) \). Therefore, when a node \( S_i \) is selected as a storage node in the optimal solution, it will always satisfy \( \Delta E(S_i) < 0 \), and its absolute value \( |\Delta E(S_i)| \) denotes the amount of energy cost reduction. For ease of representation, in the following part we also use the energy reduction \( \Delta E(S) \) to express the changed energy cost for a bunch of nodes \( S \) to switch from forwarding nodes to storage nodes. Similar to the strategy proposed in unlimited number of storages, as Fig. 4 illustrates, we divide the deployment area in irregular shape into a certain number of fan-shaped partitions. Then we compute the optimized storage placement inside each partition. The optimized storage placement relies on the following conclusion:

**Theorem 2.** In the optimal storage placement with limited number of storages, the storage nodes are all within the optimal storage area with unlimited number of storages.

**Proof.** We prove this theorem by contradiction. Suppose in the optimal storage placement with unlimited number of storages, the essential number of storages is \( K^* \). Assume in the optimal solution with \( K \) limited number of storages, there exist some storage nodes outside the boundary of the optimized storage area calculated in unlimited storage case. We denote these nodes as \( S \). Then as we increase the number of storages from \( K \) to \( K^* \), the nodes in \( S \) will obviously switch from storage nodes to forwarding nodes, being replaced by other storage nodes within the \( k_{opt} \)th hop inside each unit zone. During the process, we denote those nodes which switch from forwarding nodes to storage nodes as \( R \). Based on the above analysis, due to the optimization effect of the new storage structure, we obtain the energy reduction \( \Delta E(R) < \Delta E(S) \). Due to the increment of \( K \), we know that the size of \( R \) is larger than the size of \( S \). According to the above inequality, with \( K \) limited number of storages, we can propose a better optimized solution as compared to the current solution: switch part (or: all) of the storage nodes in \( S \) to forwarding nodes, and switch equal number of nodes in the \( R \) to be new storage nodes, we can rebuild part of the new storage structure and thus achieve better performance. Hence we get the contradiction with the former assumption, the theorem gets proved.

As indicated in Theorem 2, it can be inferred that in the optimal storage placement with limited number of storages, the storage node set is a subset of the storage nodes in the optimal storage placement with unlimited number of storages. Therefore, the solution space for the optimized storage placement is greatly reduced. Moreover, as the deployment area is divided into a certain number of fan-shaped partitions, each fan-shaped partition is fairly regular in shape, then nodes in the same hop should have the same opportunity to be storage nodes or forwarding nodes. Hence we can utilize the hop as the computation unit instead of the node, then the compute complexity is further reduced. As a matter of fact, even for a large scale sensor network, the maximum hop is very limited. Fig. 5 illustrates the coverage area of a conventional sensor network as the maximum hop count increases. Here we set the average hop distance \( d = 3m \). We note that as the maximum hop count reaches 15, the coverage area is already near 6500 \( m^2 \), which is a rather large scale deployment for conventional sensor network applications. Thus the maximum hop count is smaller than 15 for conventional cases. Therefore, with limited number of storages, it is practical to enumerate all possible storage placement solutions for any unit zone and compare their performance. The size of solution space is \( s = 2^{k_{opt}} \), as conventionally \( k_{opt} \leq 15 \), \( s \) is smaller than \( 2^{15} = 32768 \).
Based on the above analysis, if the deployment area is a circular region with the sink in the center, then due to the symmetry of the topology, there must exist a consistent value for the parameter \( k_{opt} \) to denote the boundary of the storage area. Then we enumerate all \( 2^{k_{opt}} \) solutions for the storage placement in the unit zone and maintain two tables: \( E[1..2^{k_{opt}}] \) and \( N[1..2^{k_{opt}}] \). If we use the notion item to denote a solution in a specified unit zone, then as compared to the all forwarding node solution, \( E[j] (1 \leq j \leq 2^{k_{opt}}) \) is used to hold the reduced energy for the \( j \)th item and \( N[j] (1 \leq j \leq 2^{k_{opt}}) \) is used to hold the essential number of storages for the \( j \)th item. Hence the problem is formulated as follows:

\[
\begin{align*}
\text{maximize} & \quad \sum_{j=1}^{2^{k_{opt}}} E[j] \cdot x_j \\
\text{subject to} & \quad \sum_{i=1}^{2^{k_{opt}}} N[j] \cdot x_j \leq K \\
& \quad \sum_{i=1}^{2^{k_{opt}}} x_j \cdot \theta \leq \pi
\end{align*}
\]

Here \( x_j (1 \leq j \leq k_{opt}) \) denotes the number of used copies of the \( j \)th kind of item, which actually means the number of unit zones applied with the \( j \)th solution for storage placement. The objective is to maximize the overall reduced energy cost. The first constraint means that the overall number of storage nodes should be no more than \( K \). The second constraint means that the overall number of items should be no more than \( \frac{\pi}{\theta} \), which is the number of unit zones in \( P_i \).

Obviously the above two unbounded multiple knapsack problems are all NP-hard. Therefore there exist no algorithms to correctly solve the problem in polynomial-time. Hence in the following part we propose algorithms to solve this problem in pseudo-polynomial time.

5.2 Dominance Relations

Based on the formulations in the unbounded knapsack problem, we observe that there exists such a property which can be called Dominance Relations: For a given item \( i \) in a specified partition, suppose we could find another item \( j \) such that

\[
\begin{align*}
N[j] & \leq N[i], \\
E[i] & \leq E[j],
\end{align*}
\]

and the above inequalities are satisfied with at least one less-than relationship. Then item \( i \) cannot appear in the optimal solution, since we could always improve any potential solution containing \( i \) by replacing \( i \) with \( j \). Therefore we can disregard the item \( i \) altogether. In such cases, item \( j \) is said to dominate \( i \). Therefore, solving the unbounded knapsack problem can be made easier by throwing away items that will never be needed. Hence, we can sufficiently reduce the size of search space from \( 2^{k_{opt}} \) to a certain level by leveraging the Dominance Relations.

Therefore, in order to find the inherent dominance relations, we propose an algorithm as illustrated in Algorithm1. According to the algorithm, let \( n = 2^{k_{opt}} \), the compute complexities of the sorting and the following iterations are respectively \( O(n \log n) \) and \( O(n^2) \), hence the overall compute complexity is \( O(n^2) \). In fact, due to the remove of the dominated items in time, the number of iterations can be greatly reduced. Therefore, the average compute complexity can be much less than \( O(n^2) \). Moreover, if the deployment area is divided into multiple partitions, the above algorithm needs to find the dominance relations for each particular partition \( P_i \) one by one, then the overall compute complexity is \( O(m \cdot n^2) \), here \( m \) is the number of partitions.

5.3 Dynamic Programming based Algorithm
Algorithm 1 Identify the dominance relations among the items
1: Put all $2^{k_{opt}}$ items into the search space $S$.
2: Sort the items in decreasing order of reduced energy $E_i[j]$, rename the items as $i(1 \leq i \leq 2^{k_{opt}})$ according to the rank.
3: Sort the items in increasing order of required number of storage $N_i[j]$.
4: for $i \in [1, 2^{k_{opt}}]$ and $j \in S$ do
5:   for $j \in [i + 1, 2^{k_{opt}}]$ and $j \in S$ do
6:     if $N_i[j] < N_i[j]$ then
7:       Remove item $j$ from the item set $S$, $S = S - j$.
8:   end if
9: end for
10: end for

Suppose after identifying the dominance relation, the size of the search space is reduced from $2^{k_{opt}}$ to $M$. Then we can utilize dynamic programming to solve the unbounded two-dimensional knapsack problem.

If the deployment area is a circular area with regular shape, then according to the problem formulation in (7), we maintain a table $F[j][k][l]$ to denote the maximum reduced energy cost, which can be attained with the following constraints:

- It uses items up to the $j$th kind;
- The required number of storages is less than or equal to $k$;
- The required number of items is less than or equal to $l$.

Hence we can define $f[j][k][l]$ recursively as follows:

$$
\begin{align*}
F[0][j][l] &= 0 \\
F[j][k][l] &= \max_{0 \leq x_j \leq \min(i, j/\theta)} F[j - 1][k - x_j \cdot N[j][l - x_j \cdot \theta] + x_j \cdot E[j}}
\end{align*}
$$

The above formulation shows that when $j = 0$, no items can be used, the reduced energy cost is 0; otherwise, according to the value of $F[j - 1][k - x_j \cdot N[j][l - x_j \cdot \theta]$, we can calculate the value of $F[j][k][l]$ by obtaining the maximum value of $F[j - 1][k - x_j \cdot N[j][l - x_j \cdot \theta] + x_j \cdot E[j]$ through enumerating all possible values of $x_j$.

Therefore, if we maintain a three-dimensional table $F[j][k][l]$ of size $M \times N \times \frac{M}{\theta}$, we can calculate each value of $F[j][k][l]$ recursively according to the formulation (10), then finally we can obtain the value of $f^* = F[M][K][\frac{M}{\theta}]$, which is the maximum reduced energy cost. By tracing back each $x_j(0 \leq j \leq M)$ used to achieve $f^*$, we can obtain the optimal parameters for the optimal storage placement. Since the computation of $f^*$ involves examining $M$ items, and there are $K \times \frac{M}{\theta}$ values of each $F[j][k][l]$ to calculate by using items up to $j$, the compute complexity of the dynamic programming is $O(M \cdot K \cdot \frac{M}{\theta})$.

If the deployment area is in irregular shape, then according to the problem formulation in (8), we can add an additional dimension to denote which partition the current item is put into. Then similarly we can recursively calculate the optimal solution step by step. Hence the compute complexity of the dynamic programming is $O(m \cdot M \cdot K \cdot \frac{M}{\theta})$.

5.4 Greedy Approximation Algorithm

We already know that the dynamic programming based algorithm is able to solve the problem in a pseudo-polynomial time. However, as the number of storages, $K$, increases to a large value, the compute complexity is still fairly large. Therefore, a greedy approximation algorithm is essential to be proposed to solve the above problem in a more time-efficient approach.

Consider the situation with limited number of storages, it can be understood that conventionally the storage is a scarcer resource than the other issues such as the space. Here we use the notion sack to denote the fan-shaped partition. Algorithm 2 illustrates the greedy approximation algorithm. In this algorithm, for any sack-item pair $(i, j)(1 \leq i \leq m, 1 \leq j \leq M)$, we first sort the pairs in decreasing order of reduced energy per storage unit, i.e., $E_i[j]$; then among the pairs with equivalent value of $E_i[j]$, we further sort them in decreasing order of reduced energy per degree unit, i.e., $E_i[j]$. As the degree $\theta$ is equivalent for all pairs, thus the pairs are actually sorted according to the value of $E_i[j]$.

After that, we proceed to insert the corresponding items into the sacks. Among all sack-item pairs, we start from the sack-item pair $(i, j)$ which ranks first in the sorted pairs, putting as many copies as possible of the item $j$ into the sack $i$ until there is not enough space for the item $j$ in sack $i$. Then we continue to insert the copies of the items into corresponding sacks according to the ranking of the sack-item pairs, until there is no longer space for any kind of the item in the corresponding sack for more.

Algorithm 2 Greedy Approximation Algorithm

1: Calculate $E_i[j]$ and $N_i[j]$ for all sack-item pairs $(i, j)$.
2: Sort all the sack-item pairs $(i, j)$ in decreasing order of $E_i[j]$.
3: Among the items with equivalent value of $E_i[j]$, further sort all the sack-item pairs $(i, j)$ in decreasing order of $E_i[j]$.
4: Push the ranked sack-item pairs $(i, j)$ into a queue $Q$.
5: $k = K, l_i = \frac{M}{\theta}(i \in [1, m])$
6: while $k > 0$ and $l_i > 0$ for at least one sack $i$ do
7:   Pop one sack-item pair $(i, j)$ from $Q$.
8:   if $l_i > 0$ then
9:      $x_{i,j}$ copies of item $j$ into sack $i$, where $x_{i,j} = \min(l_i, \frac{M}{\theta})$.
10:   end if
11:   $k = k - x_{i,j} \cdot N_i[j], l_i = l_i - x_{i,j} \cdot \theta(i \in [1, m])$
12: end while

The above greedy approximation algorithm has the following properties in term of performance, as shown in Theorem 3.

Theorem 3. Assume $E_{max}$ is the maximum value of items that fit into the sack in the unbounded knapsack problem, then with limited number of storages, the greedy approxima-
The greedy approximation algorithm is guaranteed to achieve at least a value of $\frac{E_{spo}}{2}$.

PROOF. In the greedy approximation algorithm, we observe that the selected item(s) must occupy at least half of the resource, i.e., the number of storages, in the sacks, otherwise, we can continue to insert the items into corresponding sacks and attain a better solution. Suppose in the worst case, the first selected item occupies more than half of the resource in any sack, then the rest part of the sacks cannot hold any other kind of items. As the reduced energy per storage unit, $E_{spo}(i,j)$, is maximized for any sack in the approximate solution, and the approximate solution can achieve the total reduced energy as $E^* \geq \frac{1}{2} \max_{i,j} E_{spo}(i,j) \cdot K$. It is apparent that in the optimal solution the total reduced energy $E_{max}$ should be less than $\frac{1}{2} \max_{i,j} E_{spo}(i,j) \cdot K$, therefore $E^* \geq \frac{1}{2} E_{max}$. The theorem gets proved.

Now consider the compute complexity of Algorithm 2. We know that the number of the sack-item pairs $(i,j)$ is $n \cdot M$, the compute complexity of the sorting is $O(mM \log(mM))$, moreover, the compute complexity of the following iteration is $O(mM)$. Therefore, the overall compute complexity of the greedy approximation algorithm is $O(mM \log(mM))$, which is irrelevant to the number of storages $K$. As $K$ can be a rather large number, the compute complexity can be greatly reduced, as compared to the compute complexity $O(m \cdot M \cdot K \cdot \frac{n}{m})$ in the dynamic programming based algorithm.

6. PERFORMANCE ANALYSIS

In this section we compare the compute complexity of our optimization methodology with former research work in [5] in respect of compute complexity.

According to the dynamic programming based algorithms depicted in [5], for the unlimited storage problem, they maintain a table $E^*[1..N]$ according to all $N$ nodes, which is used to hold the minimum energy cost of all subtrees rooted at node $i = 1, ..., N$. They also maintain a second table $E_1[1..N]$ which records the energy cost of all subtrees when all nodes in each subtree are forwarding nodes. According to their analysis the time complexity of their dynamic programming based algorithm is $O(N)$, where $N$ is the number of nodes. For the $K$ limited storage problem, they maintain a two-dimensional $(K + 1) \cdot (N - 1)$ table, $E_i[m, l]$, at each node $i$. According to their analysis, given a communication tree with $N$ nodes and at most $C$ children for each parent, the compute complexity of their dynamic programming based algorithm is $O(K \cdot N^2(max\{K, C\})(N - 1))$.

As the overall number of sensor nodes, $N$, can be a huge number for the large scale sensor network, thus the computing complexity of the above algorithms will be extremely large. Moreover, maintaining a two-dimensional table at each node is conventionally intolerable for sensors with limited resources. Since in large scale sensor networks the number of hops, $n$, is greatly smaller than the number of sensor nodes, i.e., $n \ll N$. Therefore, in this paper, we utilize the hop as the computing unit instead of the node. According to our optimized methodologies, for the unlimited storage problem, we can calculate the optimized hop-count $k_{opt}$ through Eq.(6) with constant time and space complexity, which we can denote as $O(1)$. For the limited storage problem, we reduce it into the unbounded knapsack problem, and respectively propose the dynamic programming based algorithm and the greedy approximation algorithm. The dynamic programming based algorithm maintains a four-dimensional table and the compute complexity is $O(m \cdot M \cdot K \cdot \frac{n}{m})$, here $m$ is the number of fan-shaped partitions, $M$ is the size of reduced search space, $K$ is the number of storage used, and $\theta$ is the degree of the unit zone. Moreover, the greedy approximation algorithm has a compute complexity of $O(mM \log(mM))$, which is irrelevant to the number of storages, i.e., $K$. As $K$ can be a rather large number, the compute complexity can be greatly reduced.

7. PERFORMANCE EVALUATION

We have implemented a simulator to evaluate the performance of our solutions. In the simulation, we consider a situation where sensors are uniformly deployed in a region with randomly generated shape. We randomly select a sensor node inside the deployment area as the sink. The overall number of sensor nodes is $N = 1000$. The deployment region covers an area of $6000 m^2$. Hence on average each sensor node covers an area of $6m^2$. We set the average hop distance $d = 3m$. Based on the above settings, statistically in the simulations we observe that the maximum number of hops $n$ is within the range $[10, 15]$ with 95% confidence interval. Unless otherwise stated, we set the following parameters in our simulations: query rate $r_q = 1$, query size $s_q = 1$, data generate rate $\theta_d = 1$ and data size $s_d = 1$, we set the reply reduction rate $\alpha = 0.5$, for ease of computing we set energy costs for transmitting and receiving a unit data as $e_{tx} = e_{rx} = 1$. In addition, we use relative energy cost as performance metrics. We use the case that no storage node except the sink is deployed as the baseline. Let the energy cost for this no storage scenario be $E_f$. And let the energy cost after the storage nodes are deployed be $E_s$. The relative energy cost is defined as $E_s/E_f$. In the following part we get the corresponding results by averaging the results of 1000 independent experiments.

7.1 Effect of the Optimized Placement with Unlimited Number of Storages

Fig.6(a) shows the energy cost with varying values of storage area bound, we set $r_q = 1.6$, $r_q = 1$. Here we select a specified fan-shaped partition divided from the deployment area as the test case, in which the topology is fairly regular. As the maximum number of hops in this partition is $n = 10$, we vary the storage area bound from the 1st hop to the 10th hop. We note that the energy cost starts to decrease from the results of 1000 independent experiments.
placement we randomly assign the roles (storage or forwarding) to each sensor node. We note that as $\alpha$ increases from 0.1 to 1, the energy cost of random storage placement increases from 0.33 to 1.10, and the energy cost of optimized storage placement increases from 0.22 to 1. When $\alpha = 0.1$, our optimized solution reduces 33% of the energy cost than the random storage placement; when $\alpha = 1$, our optimized storage solution can only reduce 9.1% of the energy cost than the random placement. The reason is that as the reply reduction rate increases, the transmission cost of query replies from storage nodes are increased, those storage nodes’ capability of "in network processing" cannot be sufficiently leveraged to filter enough raw sensor data to achieve the energy efficiency.

In Fig. 6(c) we compare the optimized storage placement with the random storage placement in term of the energy cost, as query rate $r_q$ varies. Note that as $r_q$ increases from 0.2 to 2, the energy cost of random storage placement increases from 0.25 to 1.2, and the energy cost of the optimized storage placement increases from 0.125 to 1. When $r_q = 0.2$, our optimized solution reduces 50% of the energy cost than the random storage placement; when $r_q = 2$, our optimized storage solution can only reduce 16.7% of the energy cost than the random placement. The reason is that as query rate increases, the query diffusion cost will become a quite large number as compared to the raw sensor data forwarding cost, thus the storage nodes’ local filtering effect to reduce energy cost will be counteracted by the large amount of query diffusion cost, hence the energy cost continue to increase as query rate increases, and more storage nodes will be replaced by forwarding nodes in the optimized solution.

### 7.2 Effect of the Optimized Placement with Limited Number of Storages

In Fig. 7, we compare various solutions in term of energy cost with varying values of $K (K < K^*)$. These solutions include the dynamic programming based algorithm, the greedy approximate algorithm and the random storage placement. In the random storage placement we randomly select $K$ nodes as the storage nodes. Among the random storage placement solutions we respectively calculate the average energy cost and the maximum energy cost for comparison. Here we set $r_q = 1.2$, $r_d = 1$. Based on the specified deployment area, we obtain the optimal number of storages $K^* \approx 700$ for the situation with unlimited number of storages. We note that as the number of storages increases, all energy costs decreases except the maximum energy cost for the random storage placement. The dynamic programming based algorithm always achieves the best performance in term of energy cost. When $K = 50$, it can reduce 16% energy cost than the average energy cost of random storage placement. As $K$ increases to 700, it can further reduce 31% energy cost than the average energy cost of random storage placement. The greedy approximate algorithm also achieves a good performance in term of energy cost, which is just next to the dynamic programming based algorithm. Moreover, among the random storage placement solutions, we observe that when $K < 150$, the maximum energy cost is contributed by the all forwarding node solution, and as $K$ increases from 150, the maximum energy cost is contributed by the solution which places all storage nodes over the last hop, which brings a large amount of query diffusion cost and reply cost.

8. CONCLUSION

In this paper, we investigate into the optimized storage placement problem over large scale sensor networks. We consider the situation with unlimited number of storages and $K$ lim-
A limited number of storages, and we leverage the characteristics of large scale sensor network to sufficiently reduce the computing complexity of our optimization methodologies. Simulation results confirm the performance of our methodologies for storage placement over large scale sensor networks.

9. ACKNOWLEDGEMENT
This work is partially supported by the National Natural Science Foundation of China under Grant No. 61100196, 61073028, 61021062; the National Basic Research Program of China (973) under Grant No. 2009CB320705; the JiangSu Natural Science Foundation under Grant No. BK2011559.

10. REFERENCES