DAO^2: OVERCOMING OVERALL STORAGE OVERFLOW IN INTERMITTENTLY CONNECTED SENSOR NETWORK

A Project

Presented

to the Faculty of

California State University, Dominguez Hills

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Hung Ngo

Fall 2020

PROJECT: OVERCOMING OVERALL STORAGE OVERFLOW IN INTERMITTENTLY CONNECTED SENSOR NETWORK.

AUTHOR: HUNG NGO

APPROVED:

_____

Project Committee Chair

_____

Committee Member

_____

Committee Member

ACKNOWLEDGEMENTS

I would like to thank all the professors that have helped and educated me in the duration of the degree program. I would also like to thank my family and friends who give me support and encourage me to pursue the master's degree.

Special thanks to Dr. Tang who have helped me and been giving advices on my master project. Thanks to Dr. Han who has given me valuable academic advices throughout my studies at CSUDH. Thanks to Dr. Beheshti's support for my academic pursuit. Thanks to all the nice people I've worked with during these two and half years.

Special thanks to my wife and my parents, who gave me unconditional love and support.

TABLE OF CONTENTS

ABSTRACT

Many emerging sensor network applications operate in challenging environments wherein sensor nodes do not always have connected paths to the base station. Data generated from such intermittently connected sensor networks therefore must be stored inside the network for some unpredictable period of time before uploading opportunities become available. Consequently, sensory data could overflow limited storage capacity available in the entire network, making discarding valuable data inevitable. To overcome such overall storage overflow in intermittently connected sensor networks, we propose and study a new algorithmic problem called data aggregation for overall storage overflow (DAO2). Utilizing spatial data correlation that commonly exists among sensory data, DAO2 employs data aggregation techniques to reduce the overflow data size while minimizing the total energy consumption.

We first study a uniform case of DAO, referred to as DAO-U, where data nodes have the same size of overflow packets and storage nodes have the same storage capacity. We uncover a new graph theoretic problem called multiple traveling salesmen selection and routing (MTSSR), and show that with proper graph transformation, the DAO-U is equivalent to the MTSSR. We prove that MTSSR is NP-hard and design a $(2 - 1/q)$ algorithm, where q is the number of nodes to visit (i.e., the number of sensor nodes that aggregate their overflow data). The approximation algorithm is based on a novel routing structure called *minimum q-edge forest* that accurately captures information needed for energy-efficient data aggregation. We further put forward a heuristic algorithm and empirically show that it constantly outperforms the approximation algorithm by 15% − 30% in energy consumption.

Then we solve the general case of DAO^2 where data nodes have varying overflow packet sizes and storage nodes have different storage capacities. DAO^2 gives rise to another new graph theoretic problem called quota multiple traveling salesmen selection and routing (Q-MTSSR). We propose an approximation algorithm for Q-MTSSR. We also propose an energy-efficient heuristic algorithms that constantly yields less cost than the approximation algorithm while collecting the same amount of prizes. We show both algorithms outperform the state-of-the-art data aggregation work that considers the availability of a base station. Finally, we propose distributed data aggregation algorithm that can achieve the same approximation ratio as the centralized algorithm under some condition, while incurring comparable energy consumption.

Keywords – Intermittently Connected Sensor Networks, Data Aggregation, Approximation Algorithms, Graph Theory

I. Introduction

In recent years sensor networks have been adopted to tackle some of the most fundamental problems facing human beings, such as disaster warning, climate change, and renewable energy. These emerging scientific applications include underwater or ocean sensor networks [33, 53], wind and solar harvesting [38], and seismic sensor networks [51]. One common characteristic of these applications is that they are all deployed in challenging environments such as in remote or inhospitable regions, or under extreme weather, to continuously collect large volumes of data for a long period of time.

In those challenging environments, it is usually not possible to deploy high-power, high storage data-collecting base stations in the field. Consequently, sensory data generated are stored inside the network for some unpredictable period of time and then collected by periodic visits of data mules [43], or by low rate satellite link [41]. We refer to such sensor networks wherein sensor nodes do not always have connected paths to the base station as *intermittently connected sensor networks*. Due to inadequate human intervention in the inhospitable environments, intermittently connected sensor networks must operate more resiliently than traditional sensor networks (with base stations and in friendly environments).

In this paper we tackle data resilience in intermittently connected sensor networks. *Data resilience* refers to the ability of long-term viability and availability of data despite insufficiencies of (or disruptions to) the physical infrastructure that stores the data. In intermittently connected sensor networks, one such disruption and major obstacle is data storage overflow. On one side, sensing a wide range of physical properties in real world, above scientific applications generate massive amounts of data, such as videos or high resolution images. On the other side, storage is still a serious resource constraint of sensor nodes despite the advances in energy-efficient flash storage [33, 39]. As a consequence, the massive sensory data could soon overflow data storage of sensor nodes and cause data loss. Such storage overflow problem is further exacerbated in intermittently connected sensor networks, wherein most of the time the high-storage base stations are not available to collect and store the data.

To avoid data loss, our previous work has designed a suite of techniques to *offload* overflow data from storage-depleted sensor nodes to nearby sensor nodes with available storages [27, 28, 45, 52]. However, if these offloaded data cannot be collected and uploaded timely by data mules or satellite links, they could soon overflow the available storage in the entire network. This unfortunately can not be alleviated by aforesaid data offloading techniques. We refer to this more severe obstacle in the intermittently connected sensor networks as *overall storage overflow*. Below we give a more concrete example contributing to overall storage overflow in intermittently connected sensor networks.

Motivation Example for Overall Storage Overflow. Consider a recent application of underwater exploration and monitoring [11]. In this applications camera sensors take pictures of the underwater

scenes while an autonomous underwater vehicle (AUV) is dispatched periodically to collect the pictures from the camera sensors. Suppose there are 100 underwater camera sensors, 10 of which are generating one 640×480 JPEG color image per second. Even using the latest parallel NAND flash technology with 16GB for sensor storage [30], it takes less than one day to exhaust the storages of all the 100 camera sensors, causing overall storage overflow. If the AUV cannot be dispatched timely due to inclement and stormy weather, discarding valuable data becomes inevitable. In this paper, we attempt to answer following question: *How to preserve the data in intermittently connected sensor networks despite the overall storage overflow?*

Fortunately, we can take advantage of spatial correlation that commonly exists among sensory data [50], and employ data aggregation techniques to reduce the overflow data size in order to overcome overall storage overflow. The spatial correlation of sensory data is due to the close proximity of sensor nodes detecting the same event of interest, thus producing data of similar values (we will provide the detailed spatial correlation data model in Section II). We formulate a new algorithmic problem called *data aggregation for overall storage overflow (DAO^2)*. At the core of DAO^2 is a new graph-theoretic problem called *multiple traveling salesmen selection and routing (MTSSR)*, which has not been studied in any existing literature. To solve DAO^2, we design a suite of energy-efficient optimal, approximation, heuristic, and distributed data aggregation algorithms with detailed analytical analysis of their performances. One novelty of our aggregation techniques is a routing structure called *minimum q-edge forest*, where q is the number of sensor nodes that aggregate their overflow data. The minimum q-edge forest generalizes minimum spanning tree, one of the most fundamental graph structures, and accurately captures information needed for energy-efficient data aggregation.

After being aggregated to the size accommodable by the network, the overflow data can then be stored into sensor nodes with available storage using data offloading techniques proposed in [27, 28, 45, 52] (we further illustrate this using Example 1 in Section II). Note that in this paper we do not consider how to upload data from sensor nodes to base station, which has been studied extensively by using data mules or mobile data collectors [22, 43]. In the conference version of this paper [44], we only considered a simplified version of DAO^2 referred to as DAO^2-U. In DAO^2-U, all the data nodes have the same size of overflow data and the storage capacity is the same for all the storage nodes.

The rest of the paper is organized as follows. Section II presents our system model and formulates DAO^2 with an illustrative example. In Section III we first study DAO^2-U. We show that DAO^2-U is equivalent to the a new graph theoretical problem called *multiple traveling salesmen selection and routing* (MTSSR), for which we design a suite of optimal, approximation and heuristic algorithms. In Section IV we study DAO^2 while identifying another new graph theoretical problem called *quota multiple traveling salesmen selection and routing* (Q-MTSSR). Again we design optimal, approximation, an heuristic algorithms to solve Q-MTSSR and DAO^2. In Section V, we design two distributed data

aggregation algorithms for the DAO^2 with time and message analyses. Section VI compare all the algorithms under different network dynamics and discuss the results. Section VII and VIII review related work and conclude the paper with possible future research.

II. Problem Formulation of DAO^2

In this section, we first introduce the DAO^2 with a problem statement. We then present its network model, data spatial correlation model, and energy model. We finally formally formulate the DAO^2 and end with an illustrative example.

Problem Statement of DAO^2. Fig. II shows an intermittently connected sensor network. In our model, some sensor nodes are close to the events of interest thus are constantly generating sensory data and have depleted their own storages. We refer to sensor nodes with depleted storage spaces while still generating data as **data nodes**. The newly generated data that can no longer be stored at data nodes is called **overflow data**. To avoid data loss, overflow data is offloaded to sensor nodes with available storages (referred to as **storage nodes**). Note that sensor nodes that have generated data but have not depleted their storage spaces are considered as storage nodes, as they can store overflow data from data nodes.

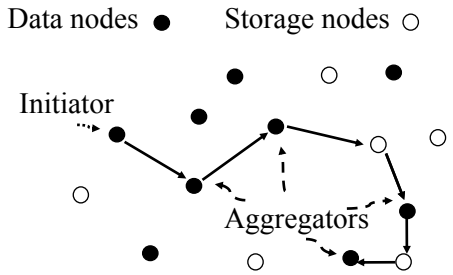


Fig. 1. An illustration of DAO^2.

To start the aggregation process, one or more data nodes (called **initiators**) send their overflow data to visit other data nodes in multi-hop manner. When a data node receives the data, it aggregates its own overflow data and becomes an **aggregator**, and then forwards the initiator's entire overflow data to another data node. This data node also becomes an aggregator and aggregates its overflow data. This continues until enough aggregators are visited such that total size of the overflow data after aggregation equals to or is slightly less than total available storage in the network. Note that during the aggregation process, it does not store overflow data to the storage nodes since other aggregators need the entire data in order to aggregate their own data.

Network Model. The sensor network is represented as an undirected connected graph $G(V, E)$, where $V = \{1, 2, ..., |V|\}$ is the set of $|V|$ sensor nodes and E is the set of $|E|$ edges. There are p data nodes, denoted as V_d (the other $|V| - p$ nodes are storage nodes), where data node $i \in V_d$ has R_i amount of

overflow data in bits. The rest $|V| - p$ sensor nodes are storage nods, where storage node $j \in V - V_d$ has m_j amount of available storage space in bits. Due to the overall storage overflow focused on this paper, $\sum_{i \in V_d} R_i > \sum_{j \in V - V_d} m_j$. Let $\mathcal{Q}$ denote the amount of data size that needs to be reduced via data aggregation; $\mathcal{Q} = \sum_{i \in V_d} R_i - \sum_{j \in V - V_d} m_j$.

Spatial Correlation Data Model. Let $H(X)$ denote the entropy of a discrete random variable X, and $H(X|Y)$ denote the conditional entropy of a random variable X given that random variable Y is known. If data node i receives no side information (i.e., overflow data) from other data nodes, its overflow data is entropy coded with $H(i|j_1, ..., j_p) = R_i$ bits, $j_k \in V_d \wedge j_k \neq i, 1 \leq k \leq p$. If data node i receives side information from at least one data node, the size of its overflow data is $H(i|j_1, ..., j_p) = r_i \leq R_i$. This correlation model has two advantages. First, it captures the uniform data spatial correlation scenario, wherein data generated at different data nodes have similar correlation with each other (we leave the more challenging and realistic model that different nodes have different data correlation as future work). Second, it is an effective distributed coding strategy, which works well in large scale sensor network applications. We are aware of other distributed coding techniques such as Slepian-Wolf coding [54]. However, they need global correlation structure, which is impractical in large networks.

Correlation Coefficient. Based on above spatial correlation model, we further define *correlation coefficient* as the percentage of data size that can be reduced at each data node and denote it as ρ. That is, $\rho = 1 - r_i/R_i$, $i \in V_d$, indicating that $R_i \cdot \rho$ amount of overflow data at data node i can be reduced via aggregation. ρ shows the similarities thus redundancies of data on different data nodes. $\rho = 0$ means no spatial correlation at all thus data at data nodes are totally different from each other and cannot be aggregated, while $\rho = 1$ means perfect correlation where data at aggregators are duplicate copies of data at initiators thus can be completely removed. Given any instance of overall storage overflow where $\sum_{i \in V_d} R_i > \sum_{j \in V - V_d} m_j$, a specific ρ value is called effective if $\rho \times \sum_{i \in V_d} R_i \geq \sum_{i \in V_d} R_i - \sum_{j \in V - V_d} m_j$, thus $\rho \geq \frac{\sum_{i \in V_d} R_i - \sum_{j \in V - V_d} m_j}{\sum_{i \in V_d} R_i}$. Denote *threshold correlation coefficient* as $\rho_{th} = \frac{\sum_{i \in V_d} R_i - \sum_{j \in V - V_d} m_j}{\sum_{i \in V_d} R_i}$; ρ_{th} is the minimum effective ρ that solves an overall storage overflow instance.

Our model is based on a well-known entropy-based model proposed in [19], with one difference. The model in [19] assumes that each sensor node generates some data packets and transmits them back to the base station. In our model, we only consider overflow data packets generated only at data nodes and the goal is to reduce their sizes instead of transmitting them back to the base stations. We have two observations about the correlation model.

Observation 1: Each data node can be either an initiator, or an aggregator, or none of them, but not both of them. An initiator cannot be an aggregator because its data serves as side information for other nodes to aggregate. An aggregator cannot be an initiator since its aggregated data loses the side information needed for others nodes' aggregation. □

Observation 2: Each aggregator can be visited multiple times by the same or different initiators (if

that is more energy-efficient). However, the data of an aggregator i can only be aggregated once, with size reduced from R_i to r_i. □

Energy Model. We adopt first order radio model [25] for battery power consumption. When node u sends R_u-bit data to its one-hop neighbor v over distance $l_{u,v}$, *transmission cost* at u is $E_t(R_u), l_{u,v}) = E_{elec} \times R_u + \epsilon_{amp} \times R_u \times l_{u,v}^2$, *receiving cost* at v is $E_r(R_u) = E_{elec} \times R_u$. Here, $E_{elec} = 100nJ/bit$ is energy consumption per bit on transmitter and receiver circuits, and $\epsilon_{amp} = 100pJ/bit/m^2$ is energy consumption per bit on transmit amplifier. Let $W = \{v_1, v_2, ..., v_n\}$ be a *walk*, a sequence of n nodes with $(v_i, v_{i+1}) \in E$ and $v_1 \neq v_n$ (if all nodes in W are distinct, W is a *path*). Let $w(R_u, u, v) = E_t(R_u, l_{u,v}) + E_r(R_u)$, and $c(R_u, W) = \sum_{i=1}^{n-1} w(R_u, v_i, v_{i+1})$ denote the *aggregation cost* on W, the energy consumption of sending R_u-bit from v_1 to v_n along W. We assume that there exists a contention-free MAC protocol to avoid overhearing and collision (e.g. [14]).

Problem Formulation of DAO^2. DAO^2 determines a set of a ($1 \leq a < p$) initiators, denoted as $\mathcal{I}$, and corresponding set of a *aggregation walks/paths*: $\mathcal{W} = \{W_1, W_2, ..., W_a\}$, where W_j ($1 \leq j \leq a$) starts from a distinct initiator $I_j \in \mathcal{I}$. Let G_j be the set of storage nodes in W_j thus $W_j - \{I_j\} - G_j$ is the set of aggregators in W_j. Let A denote all the aggregators that are visited in $\mathcal{W}$; $A = \bigcup_{j=1}^{a} \{W_j - \{I_j\} - G_j\}$. The goal of DAO^2 is to find $\mathcal{W}$ and $\mathcal{I}$ such that the total amount of reduced data $\sum_{i \in A} (R_i \cdot \rho) >= \mathcal{Q}$ while minimizing the *total aggregation cost* $\sum_{1 \leq j \leq a} c(R_{I_j}, W_j)$, the total energy consumption incurred in the aggregation process. Note that $A = \bigcup_{j=1}^{a} \{W_j - \{I_j\} - G_j\}$ guarantees that even though an aggregator can be visited multiple times (by the same or different initiators), its data reduction is only counted once. Table I lists all the notations for above and later sections.

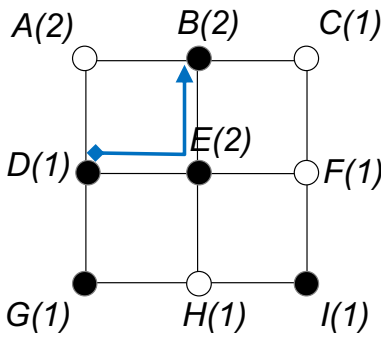


Fig. 2. An example for DAO^2. Numbers are R_i (for data nodes ●) and m_j (for storage nodes ○). Blue arrowed line shows the aggregation path.

EXAMPLE 1: Fig. II gives an example of DAO^2 in a grid sensor network of 9 nodes (we use grid only for illustration purpose). Nodes B, D, E, G, and I are data nodes, with $R_B = R_E = 2$ while $R_D = R_G = R_I = 1$. Nodes A, C, F and H are storage nodes, with $m_j = 1$ for all of them except that $m_A = 2$. The energy consumption along any edge is 1 for one unit of data and correlation coefficient

TABLE I

NOTATIONS USED IN DIFFERENT PROBLEMS

DAO^2	
$G(V, E)$	Sensor network where DAO^2 and DAO^2-U run
V_d and p	Set and number of data nodes
m_i	Storage capacity of a storage node $i \in V - V_d$
R_i	Overflow data size at data node i before aggregation
r_i	Overflow data size at data node i after aggregation
ρ	Correlation coefficient, $\rho = 1 - r_i/R_i$, $i \in V_d$
$\mathcal{Q}$	$\mathcal{E} = \sum_{i \in V_d} R_i - \sum_{i \in V - V_d} m_i$
$\mathcal{I}$, a	Set and number of initiators, $1 \leq a \leq (p - q)$
I_j	j^{th} initiator, $1 \leq j \leq a$
W_j	Aggregation walk or path starting with I_j
$w(R_u, u, v)$	Aggr. cost of sending R_u bits from u to neighbor v
$c(R_u, W_j)$	Aggr. cost of sending R_u bits along W_j
DAO^2-U	
q	Number of aggregators needed
m	Storage capacity of a storage node in $V - V_d$
R	Overflow data size at each data node before aggregation
r, $r < R$	Overflow data size at each data node after aggregation
MTSSR and Q-MTSSR	
$G(V', E')$	Aggregation network where MTSSR and Q-MTSSR run
$w(u, v)$	Weight of an edge $(u, v) \in E'$
pr_i	Prize at node $i \in V'$
$cost_i$	Weight of node $i \in V'$
$\mathcal{Q}$	Total amount of prizes to be collected
$pcr(C_i, C_j)$	Prize-cost ratio of two components C_i and C_j
$\mathcal{B}(u, v)$	Benefit of an edge $(u, .v) \in E'$, $\mathcal{B}(u, v) = \frac{pr_u + pr_v}{w(u,v)}$
$\mathcal{T}$	Total traveling cost in MTSSR and Q-MTSSR

$\rho = 1/2$. Overall storage overflow exists as there are 7 units of overflow data but only 5 units of storage spaces, giving $\mathcal{Q} = 2$. The optimal solution, shown in blue arrowed line, is selecting D as the initiator and setting its aggregation path as: D, E, and B, with the total aggregation cost of 2. After aggregation, the sizes of overflow data at B, E, D, G, and I are 2, 1, 0, 1, 1 respectively, totaling five units. Note that 2 units of data at B now include 1 unit of B's own aggregated data and 1 unit of initiator D's intact data. Now the five units of overflow data can fit and be stored into the five units of available storage spaces, solving the overall overflow problem. □

Data Offloading After Data Aggregation. After aggregation, the next question is how to offload data from data nodes to storage nodes with minimum energy consumption. Our previous work [28, 45] have shown this can be modeled as a minimum cost flow problem [1], which can be solved optimally and

efficiently. One optimal solution in Fig. II is offloading the 2 units of data at B to A, E's 1 unit of aggregated data to C, G's 1 unit of intact data to H, and I's own 1 unit of intact data to F, totaling 6 offloading cost.

As data offloading can be achieved optimally, we only focus on data aggregation. In Section III we consider a uniform scenario where all the data nodes have the same overflow data size (i.e., $R_i = R$) and all the storage nodes have the same storage capacity (i.e., $m_j = m$). We refer to it as DAO^2-U. In Section IV we study the general DAO^2 with heterogenous R_i and m_j. In this paper data aggregation and data offloading are treated as separated stages; an integrated, more energy-efficient solution was proposed in [3].

III. Algorithmic Solutions for DAO^2-U

A. Problem Formulation of DAO^2-U.

When $R_i = R$, $i \in V_d$ and $m_j = m$, $j \in V - V_d$, the overall storage overflow condition becomes $p \times R > (|V| - p) \times m$, giving that $p > \frac{|V|m}{m+R}$. We are able to calculate the number of aggregators, denoted as q, that need to be visited for the data reduction. Since each aggregator reduces its overflow data size by $(R-r)$, and the total anticipated data size reduction is $p \times R - (|V| - p) \times m = p \times (R+m) - |V| \times m$, we have

$$q = \lceil \frac{p \times (R+m) - |V| \times m}{R - r} \rceil. \tag{1}$$

To guarantee that the overflow data after aggregation can fit in the available storage in the network, next we compute the upper bound of number of data nodes p. As at least one data node needs to be the initiator to start the aggregation process, there can only be maximum of $p-1$ aggregators (Observation 1). We therefore have $q = \lceil \frac{p \times (R+m) - |V| \times m}{R-r} \rceil \leq p - 1$, which gives $p \leq \lfloor \frac{|V|m - R + r}{m+r} \rfloor$. As we have calculated the lower bound of p in network model above, the valid range of p for overall storage overflow to occur is therefore

$$\frac{|V|m}{m+R} < p \leq \lfloor \frac{|V|m - R + r}{m + r} \rfloor. \tag{2}$$

Given a valid p value and its corresponding q value, meaning q out of the p data nodes need to be aggregators and the rest $p - q$ data nodes can serve as initiators (Observation 1), DAO^2-U determines:

- set of a ($1 \leq a \leq (p - q)$) initiators, denoted as $\mathcal{I}$, and
- corresponding set of a *aggregation walks*: W_1, W_2, ..., W_a, where W_j ($1 \leq j \leq a$) starts from a distinct initiator $I_j \in \mathcal{I}$, such that $|\bigcup_{j=1}^{a} \{W_j - \{I_j\} - G_j\}| = q$. Here, G_j is the set of storage nodes along W_j thus $W_j - \{I_j\} - G_j$ is the set of aggregators in W_j. Since an aggregator can appear multiple times in the same or different aggregation walks (Observation 2), $\bigcup_{j=1}^{a} \{W_j - \{I_j\} - G_j\}$ signifies a set of q *distinct* aggregators in the network.

The goal DAO^2-U is to minimize the *total aggregation cost* $\sum_{1 \leq j \leq a} c(R, W_j)$, the total energy consumption incurred in the aggregation process.

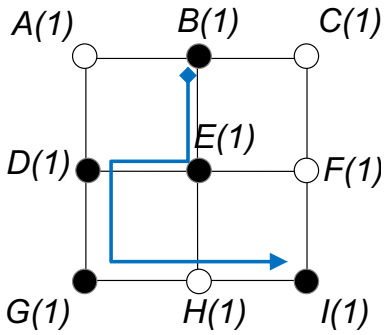


Fig. 3. An example for DAO^2-U. Numbers are R_i (for data nodes •) and m_j (for storage nodes ○). Blue arrowed line shows the aggregation path.

EXAMPLE 2: Fig. III-A shows the same sensor network in Fig. II with the same sets of data and storage nodes. Now assume that $R = m = 1$ for the uniform case and $\rho = 3/4$. Again overall storage overflow exists as there are 4 units of storage space while there are 5 units of overflow data. Using Equation 1, the number of aggregators q is 4, leaving one data node to be initiator. One of the optimal solutions is selecting B as initiator and setting its aggregation path as: B, E, D, G, H, I, as shown in the blue arrowed line, with total aggregation cost of 5. After aggregation, the sizes of overflow data at B, E, D, G, and I are 0, 3/4, 3/4, 3/4, and 7/4, respectively, which is total 4 units thus can be offloaded to storage nodes using techniques in [28, 45]. Note that 7/4 units of data at I now include 3/4 unit of I's own aggregated overflow data and one unit of initiator B's overflow data. □

We find that DAO^2-U gives rise to a new graph-theoretic problem, which we refer to as *multiple traveling salesmen selection and routing (MTSSR)*. In Section III-B we formulate MTSSR, prove its NP-hardness, and solve it by an efficient $(2 - \frac{1}{q})$-approximation algorithm. Section III-C then proves that the DAO^2-U in a sensor network is equivalent to the MTSSR in a so-called aggregation network transformed from the sensor network, therefore the algorithms for MTSSR can be applied to solve DAO^2-U.

B. Multiple Traveling Salesmen Selection and Routing (MTSSR)

1) Problem Formulation and NP-Hardness.: Given an undirected weighted graph $G' = (V', E')$ with $|V'|$ nodes and $|E'|$ edges, a cost metric that represents the distance or traveling time between two adjacent nodes, and that the number of nodes that must be visited is q. The objective of the MTSSR is to determine a set of *at most* $|V'| - q$ *starting nodes*, from each of which a salesman is dispatched to visit some nodes following a walk, such that a) all together q nodes are visited, and b) total cost of the walks is minimized.

Let $w(u, v)$ denote weight of edge $(u, v) \in E'$. We assume that triangle inequality holds: for edges $(x, y), (y, z), (z, x) \in E$, $w(x, y) + w(y, z) \geq w(z, x)$. Given a walk $W = \{v_1, v_2, ..., v_n\}$, let $c(W) = \sum_{i=1}^{n-1} w(v_i, v_{i+1})$ denote its cost. The objective of MTSSR is to decide:

- the set of a ($1 \leq a \leq |V'| - q$) starting nodes $\mathcal{I} \subset V'$, and
- the set of a *walks* $W_1, W_2, ..., W_a$: W_j ($1 \leq j \leq a$) starts from a distinct node $I_j \in \mathcal{I}$, and $|\bigcup_{j=1}^{a}\{W_j - \{I_j\}\}| = q$,

such that *total cost* $\sum_{1 \leq j \in a} c(W_j)$ is minimized.

Theorem 1: The MTSSR is NP-hard.

Proof: Given an undirected graph $G'(V', E')$, its *metric completion*, denoted as $G^{mc}(V, E^{mc})$, is a complete graph with the same set of nodes V', while for any pair of nodes $u, v \in V'$, the cost of $(u, v) \in E^{mc}$ is the cost of the shortest path connecting u and v in $G'(V', E')$. Recall that *traveling salesman path problem (TSPP)* [26] is to find a minimum-cost *hamiltonian path* that visits each node exactly once in a complete graph. MTSSR in $G'(V', E')$ is thus a *multiple traveling salesman path (MTSPP)* problem in $G^{mc}(V', E^{mc})$. MTSPP selects at most b starting nodes, from each of which a salesman is dispatched to visit a distinct subset of nodes following one of its hamiltonian paths, such that *exactly* q other nodes are visited with minimum total cost. We therefore prove MTSPP in $G^{mc}(V', E^{mc})$ is NP-hard. In particular, we prove TSPP, a special case of MTSPP, is NP-hard. Below we reduce the well-known *traveling salesman problem (TSP)* [18] to TSPP. Recall that TSP is to find a minimum-cost *hamiltonian cycle* in a complete graph that visits each node exactly once.

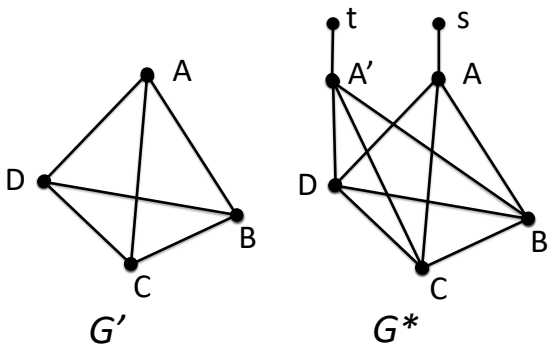
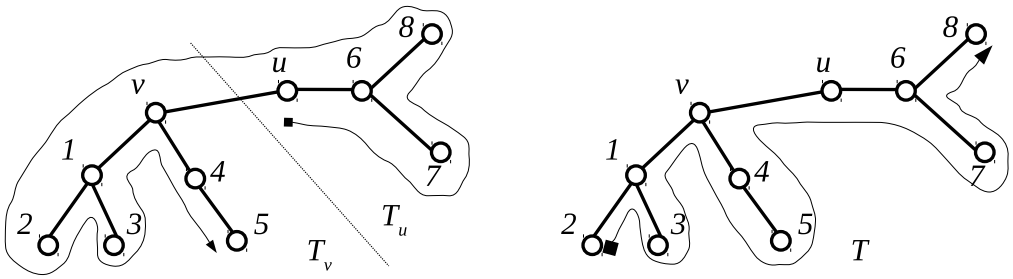


Fig. 4. Proving TSPP is NP-hard.

As shown in Fig. III-B1, let complete graph G' be an instance of TSP, we construct an instance of TSPP, G^*, as follows. We choose an arbitrary node A in G' and add a copy of it, $A^{'}$. We connect A' to all other nodes except A, and assign the same cost on each edge as the corresponding edge in G' (that is, $(A^{'}, B)$ has the same cost as (A, B), $(A^{'}, C)$ has the same cost as (A, C), etc.). Then we introduce nodes s and t and add edges (s, A) and $(t, A^{'})$ with any *finite* edge costs. Finally, as G^* must be a complete graph, we add the rest edges (not shown in Fig. III-B1) and assign their costs to be *infinite*. We show that G contains a minimum-cost Hamiltonian cycle, say, A, C, B, D, A, if and only if G^* contains a minimum-cost Hamiltonian path s, A, C, B, D, $A^{'}$, t.



(a) B-Walk. (b) LP-Walk.

Fig. 5. (a) A binary walk (B-walk): u, 6, 7, 6, 8, 6, u, v, 1, 2, 1, 3, 1, v, 4, 5, with cost of 16. (b) A longest-path walk (LP-Walk): $2, 1, 3, 1, v, 4, 5, 4, v, u, 6, 7, 6, 8$, with cost of 14. ■ and ◄– indicate the first and last node in a walk, respectively. Here, $w(u, v) = 2$ and weights of other edges are 1.

Suppose that G' contains a minimum-cost Hamiltonian cycle A, C, B, D, A. Then we get a minimum-cost Hamiltonian path in G^* when we start from s, follow the cycle back to $A^{'}$ instead of A, and finally end in t. Conversely, suppose G^* contains a minimum-cost Hamiltonian path. This path (with finite cost) must end in s and t. We transform it to a cycle in G' by a) deleting s and t, which results in a path that end in A and $A^{'}$, and b) removing $A^{'}$. The resulted path, instead of going back to $A^{'}$, goes back to A, forming a minimum-cost Hamiltonian cycle in G'. ■

2) Approximation Algorithm for MTSSR: We first introduce some definitions.

Definition 1: (**Binary Walk (B-Walk).**) Given a tree $T \subset G'$ with a maximum-weight edge (u, v) (ties are broken randomly), T is divided into (u, v) and subtrees T_u and T_v. The B-walk on T, denoted as $W_B(T)$, starts from u and visits all the nodes in T_u following depth-first-search (DFS), and then visits v, from where it visits all the nodes in T_v following DFS and stops when all the nodes are visited. □

Fig. 5(a) shows a tree T with $w(u, v) = 2$ and weights of other edges being 1, and a B-walk of cost 16. In B-walk, each edge in T_u is traversed twice, and each edge in T_v is traversed once or twice. B-walk saves cost traversing a tree since the maximum-weight (u, v) is traversed only once.

Lemma 1: $c(W_B(T)) \leq (2 - \frac{1}{|T|}) \times c(T)$. Here $c(T) = \sum_{e \in T} w(e)$ and $|T|$ is the number of edges in T.

Proof: Since (u, v) is the edge in T with maximum weight, $w(u, v) \geq \frac{1}{|T|} \times c(T)$. In $W_B(T)$, since (u, v) is traversed exactly once and other edges are traversed *at most* twice, $c(W_B(T)) \leq \big(2 \times c(T) - w(u, v)\big)$. Therefore $c(W_B(T)) \leq \big(2 \times c(T) - \frac{1}{|T|} \times c(T)\big) = (2 - \frac{1}{|T|}) \times c(T)$. ■

Definition 2: (**Forest and q-Edge Forest**) A forest F of G' is a subgraph of G' that is acyclic (and possibly disconnected). A q-edge forest F_q is a forest with q edges. □

Approximation Algorithm. Algo. 1 works as follows. Line 1 and 2 sort all the edges in E' in non-descending order of their weights, and initialize an edge set E_q to be empty. The while loop in lines

3-9 finds the first q edges in E' that do not cause a cycle and store them in E_q. It then obtains a q-edge forest $G'[E_q]$ (line 10). Each connected component of $G'[E_q]$ is either linear or a tree as no cycles are introduced. If it is linear, it starts from one end and visits the rest nodes exactly once; if it is a tree, it does a B-walk to visit all the nodes (lines 11-15).

Algorithm 1: Approximation Algorithm for MTSSR.

Input: $G'(V', E')$ and number of nodes to visit q;

Output: a walks: $W_1, W_2, ..., W_a$, and $\sum_{1 \leq j \in a} c(W_j)$;

0. **Notations**:
   E_q: set of q cycleless edges;
   $G'[E_q]$: a q-edge forest;
   $C(G'[E_q])$: set of connected components in $G'[E_q]$;
   C_j: the j^{th} connected component in $C(G'[E_q])$;
1. Let $w(e_1) \leq w(e_2) \leq ... \leq w(e_{|E|})$;
2. $E_q = \phi$ (empty set), $i = j = k = 1$;
3. **while** $(k \leq q)$
4. **if** (e_i is a cycleless edge w.r.t. E_q)
5 $E_q = E_q \cup \{e_i\}$;
6. $k++$;
7. **end if;**
8. $i++$;
9. **end while;**
10. Let $|C(G'[E_q])| = a$; /*a connected components*/
11. **for** $(1 \leq j \leq a)$
12. **if** (C_j is linear) Start from one end node of C_j and visit the rest nodes in C_j once;
13. **if** (C_j is a tree) Do a B-walk on C_j;
14. Let the resulted walk (or path) be W_j;
15. **end for;**
16. **RETURN** $W_1, W_2, ..., W_a$, and $\sum_{1 \leq j \in a} c(W_j)$.

Discussions. Algo. 1 takes $O(|E'|\log|E'|)$ and works alike the well-known Kruskal's minimum spanning tree (MST) algorithm [17], except that instead of finding $|V'| - 1$ edges to connect all the nodes in V', it finds $q \leq |V'| - 1$ edges to "connect" *some* nodes in V'. Algo. 1 therefore generalizes Kruskal's MST algorithm, as MST is a special case of $G'[E_q]$, which is the *minimum q-edge forest* formally defined below.

Definition 3: (**Minimum q-Edge Forest**) Let $c(F_q) = \sum_{e \in F_q} w_e$ denote the cost of a q-edge forest F_q in G'. Let $\mathcal{F}_q$ be the set of all q-edge forests in G'. A q-edge forest F_q^m is minimum iff $c(F_q^m) \leq c(F_q), \forall F_q \in \mathcal{F}_q$. □

Lemma 2: $G'[E_q]$ is a minimum q-edge forest.

Proof: Let $E' = \{e_1, e_2, ..., e_{|E|}\}$, with $w(e_1) \leq w(e_2) \leq ... \leq w(e_{|E|})$. Let $E_q = \{e_1^g, e_2^g, ..., e_q^g\}$, with $w(e_1^g) \leq w(e_2^g) \leq ... \leq w(e_q^g)$. By way of contradiction, assume that another q-edge forest, O_q, is a minimum q-edge forest with cost smaller than that of $G'[E_q]$. Let $O_q = \{e_1^o, e_2^o, ..., e_q^o\}$ with $w(e_1^o) \leq w(e_2^o) \leq ... \leq w(e_q^o)$. Assume that $e_l^g \in E_q$ and $e_l^o \in O_q$, $1 \leq l \leq q$, are the first pair of edges that differ in E_q and O_q: $e_l^g \neq e_l^o$ and $e_i^g = e_i^o$, $\forall\ 1 \leq i \leq l-1$. According to Algo. 1, $w(e_l^g) \leq w(e_l^o)$. Now consider subgraph $O_q \cup \{e_l^g\}$.

Case 1: $O_q \cup \{e_l^g\}$ is a forest. Then $c(O_q \cup \{e_l^g\} - \{e_l^o\}) \leq c(O_q)$, contradicting that O_q is a minimum q-edge forest.

Case 2: $O_q \cup \{e_l^g\}$ is not a forest, i.e., there is a cycle in it. e_l^g must be in this cycle since there is no cycle in O_q. Besides, among all the edges in this cycle that is not e_l^g, at least one of them is not in E_q; otherwise there will not be any cycle (as they all belong to E_q, which is cycleless). Denote this edge as e'. Let e_l^g be the n^{th} edge in $E' = \{e_1, e_2, ..., e_{|E'|}\}$, that is, $e_l^g = e_n, 1 \leq n \leq |E'|$.

Case 2.1: $e' \in \{e_1, e_2, ..., e_{n-1}\}$. Thus $w(e') \leq w(e_{n-1}) \leq w(e_n) = w(e_l^g) \leq w(e_l^o)$, contradicting that e_l^g and e_l^o are the first pair of edges that differ in E_q and O_q.

Case 2.2: $e' \in \{e_{n+1}, e_{n+2}, ..., e_{|E|}\}$. Thus $w(e') \geq w(e_{n+1}) \geq w(e_n) = w(e_l^g)$. $c(O_q \cup \{e_l^g\} - \{e'\}) \leq c(O_q)$, contradicting that O_q is a minimum q-edge forest.

Reaching contradiction in all the cases, it concludes that $c(G[E_q]) \leq c(F_q), \forall F_q \in \mathcal{F}_q$. ∎

Let O be an optimal algorithm of MTSSR with minimum cost of $\mathcal{O}$. Next we show $c(G'[E_q])$ is a lower bound of $\mathcal{O}$.

Lemma 3: $c(G'[E_q]) \leq \mathcal{O}$.

Proof: Assume that all the edges selected in O induce λ connected components, denoted as O_j ($1 \leq j \leq \lambda$). Assume that there are l_j nodes in O_j, and s_j ($l_j > s_j \geq 1$) of them are starting nodes (therefore there are s_j walks in O_j visiting altogether $l_j - s_j$ nodes). Denote the s_j walks in O_j as W_j^o and let $c(W_j^o)$ be its cost. We have $\sum_{j=1}^{\lambda} c(W_j^o) = \mathcal{O}$.

Let $c(O_j) = \sum_{e \in O_j} w(e)$. Denote any spanning tree of O_j as T_j^o, and let $c(T_j^o) = \sum_{e \in T_j^o} w(e)$. We have $c(T_j^o) \leq c(O_j) \leq c(W_j^o)$. The first inequality is because all the edges in T_j^o are in O_j (but not vice versa); the second inequality is because each edge in O_j is traversed at least once in O. Therefore $\sum_{j=1}^{\lambda} c(T_j^o) \leq \sum_{j=1}^{\lambda} c(W_j^o) = \mathcal{O}$.

Let $q' = \sum_{j=1}^{\lambda} |T_j^o|$, where $|T_j^o|$ is the number of edges in T_j^o. We have $q' = \sum_{j=1}^{\lambda}(l_j - 1)$. The subgraph induced by all T_j^o ($1 \leq j \leq \lambda$) is therefore a q'-edge forest. Since all together q nodes are visited, $\sum_{j=1}^{\lambda}(l_j - s_j) = q$. Since $s_j \geq 1$, we have $q \leq \sum_{j=1}^{\lambda}(l_j - 1) = q'$. Therefore, $c(G[E_q]) \leq$

$c(G[E_{q'}]) \overset{\text{Lemma 2}}{\leq} \sum_{j=1}^{\lambda} c(T_j^o) \leq \mathcal{O}$. ∎

Theorem 2: Algo. 1 is a $(2 - \frac{1}{q})$-approximation algorithm.

Proof: In Algo. 1, each of the a connected components C_j $(1 \leq j \leq a)$ is either linear or a tree. Let q_j and $c(C_j)$ denote the number of edges in C_j and the sum of weights of edges in C_j, respectively. We have $q = \sum_{j=1}^{a} q_j$ and $c(G[E_q]) = \sum_{j=1}^{a} c(C_j)$. Let W_j be a B-DFS walk of C_j.

$$\begin{aligned}
\sum_{j=1}^{a} c(W_j) &\overset{\text{Lemma 1}}{\leq} \sum_{j=1}^{a} \left((2 - \frac{1}{q_j}) \times c(C_j) \right) \\
&< \sum_{j=1}^{a} \left((2 - \frac{1}{q}) \times c(C_j) \right) \\
&= (2 - \frac{1}{q}) \times c(G[E_q]) \\
&\overset{\text{Lemma 3}}{\leq} (2 - \frac{1}{q}) \times \mathcal{O}.
\end{aligned}$$

∎

Corollary 1: If C_j $(1 \leq j \leq a)$ resulted from Algo. 1 are all linear, Algo. 1 is optimal.

Proof: In this case, $\sum_{1 \leq j \in a} c(W_j) = \sum_{1 \leq j \in a} c(C_j) = c(G[E_q]) \overset{\text{Lemma 3}}{\leq} \mathcal{O}$. Since $\sum_{1 \leq j \in a} c(W_j) \geq \mathcal{O}$, we have $\sum_{1 \leq j \in a} c(W_j) = \mathcal{O}$. ∎

Smaller-Tree-First-Walk (STF-Walk). When a B-Walk traverses T_u first and then T_v, each edge in T_u is traversed twice while each edge in T_v is traversed once or twice. A simple improvement is to traverse, between T_u and T_v, the one with smaller cost first. We refer to this as *smaller-tree-first-walk (STF-walk)*. The walk in Fig. 5(a) is indeed an STF-walk.

A Heuristic Algorithm. Next we present another heuristic algorithm. It differs with Algo. 1 only in line 13: Instead of a B-walk along each tree, it does a *longest-path walk*.

Definition 4: (**Longest-Path Walk (LP-Walk).**) Let $P = \{v_1, v_2, ..., v_n\}$ be a longest path in tree T. A *LP-walk* starts from v_1, visiting all the nodes in T following DFS, and ends at v_n, such that every edge in P is traversed once. □

In LP-walk, since more edges are traversed only once, the cost of a walk can be further reduced. Finding longest path in a tree is to find the shortest path among all pair of leaf nodes and choose the longest one, which takes $O(|V|^3)$. Fig 5(b) shows a LP-walk with cost of 14. Because the maximum-weight edge (u, v) is not necessarily on the longest path P, we are not able to obtain performance guarantee for LP-walk. However, we show empirically in Section VI that it outperforms Algo. 1 by $15\% - 30\%$ in terms of energy consumption under different network parameters.

C. Equivalency Between MTSSR and DAO^2-U

Now we transform the original sensor network $G(V, E)$ into an *aggregation network* $G'(V', E')$, which is defined below, and prove that solving DAO^2-U in G is equivalent to solving MTSSR in G'.

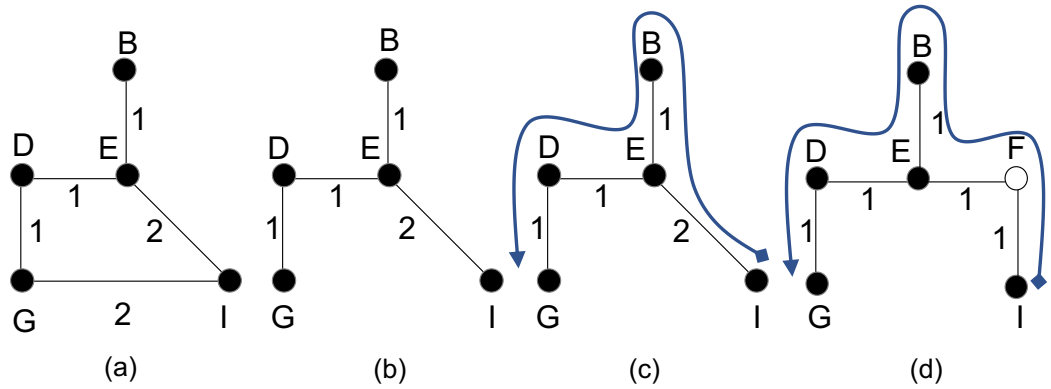


Fig. 6. (a) Aggregation network G' of sensor network G in Fig. III-A. (b) 4-edge forest F_q resulted from Algo. 1. (c) B-walk on F_q. (d) Aggregation walk in G with aggregation cost of 6. The numbers on edges are their weights.

Definition 5: (**Aggregation Network** $G'(V', E')$) V' is the set of p data nodes in V, $V' = V_d$. For any two data nodes $u, v \in V'$, there exists an edge $(u, v) \in E'$ if and only if all the shortest paths between u and v in G do not contain any other data nodes. For edge $(u, v) \in E'$, its weight $w(u, v)$ is the cost of the shortest path between u and v in G. □

EXAMPLE 3: Fig. 6(a) shows the aggregation network G' of sensor network G in Fig. II. Fig. 6(b) shows a 4-edge forest F_q of G' from Algo. 1. Fig. 6(c) shows the B-walk on F_q. Fig. 6(d) shows the aggregation walk in G by replacing each edge (u, v) in F_q with a shortest path between u and v in G. The total aggregation cost following this walk is 6, one more than the optimal cost shown in Example 2. The B-walk in this example happens to be a LP-walk. □

Implementation. One challenge of generating aggregation network G' from sensor network G is to find all the shortest paths between two data nodes u and v in G.

Theorem 3: DAO^2-U in G is equivalent to MTSSR in G'.

Proof: To prove their equivalence, it suffices to show that the data, energy consumption, and topology information that are used for computing energy-efficient aggregation in $G(V, E)$ are all preserved in $G'(V', E')$. Below we show that this is achieved during the construction of G' from G.

First, as all the data nodes in G are now nodes in G', data information is preserved. Second, if all the shortest paths between a pair of data nodes X and Y do not contain any other data nodes, then in G' all those shortest paths are replaced by one single edge (X, Y), whose weight is the cost of any of such shortest paths. Therefore the energy consumption information is preserved. Third, if there exists at least one shortest path between data nodes X and Y in G that includes at least another data node, there is no edge (X, Y) in G'. This mandates that if X and Y participate in the same aggregation walk, they should take a shortest path between them with data nodes as intermediate nodes as part of the aggregation walk. Therefore the topological requirement of DAO^2-U to "visit as many data nodes (aggregators) as possible while using as least amount of energy as possible" is preserved (Otherwise,

less number of aggregators are visited with the same amount of energy consumption, which is against the goal of DAO^2-U). Therefore, solving MTSSR in G' is equivalent to solving DAO^2-U in G. ∎

IV. Algorithmic Solutions for DAO^2

In this section, we solves the general problem of DAO^2. We show that it gives rises to another new graph theoretical problem, which we refer to as *Quota Multiple Traveling Salesmen Selection and Routing* (Q-MTSSR). Q-MTSSR generalizes MTSSR studied in Section III-B thus is NP-hard. Below we formulate Q-MTSSR, briefly show its equivalency to DAO^2, and propose approximation and heuristic algorithms.

A. Quota Multiple Traveling Salesmen Selection and Routing (Q-MTSSR)

Problem Formulation. In an undirected weighted graph $G' = (V', E')$ with $|V'|$ nodes (or cities) and $|E'|$ edges, $w_{u,v}$ is the weight on edge $(u, v) \in E'$ (indicating the distance from u to v), node $i \in V'$ has a prize pr_i to be collected, and $\mathcal{Q}$ is the targeted quota to collect.[1] Besides, node i has a weight $cost_i$ indicating a salesman's traveling cost per unit distance if he is dispatched from i.[2] Given a walk $W = \{v_1, v_2, ..., v_n\}$, the *traveling cost* on W by salesman dispatched from node i is $c(i, W) = cost_i \cdot \sum_{j=1}^{n-1} w(v_j, v_{j+1})$. The objective of the Q-MTSSR is to determine a) a set of *starting nodes* $\mathcal{I} \subset V'$, from $I_j \in \mathcal{I}$ a salesman is dispatched, and b) a walk W_j along which a sequence of nodes he visits, s.t. *total traveling cost* $\mathcal{T} = \sum_{1 \le j \in |\mathcal{I}_|} c(I_j, W_j)$ is minimized and *total collected prizes* $\sum_{k \in A} pr_k \ge \mathcal{Q}$. Here $A = \bigcup_{j=1}^{|\mathcal{I}|} \{W_j - \{I_j\}\}$, which guarantees that each node's prize is collected at most once. Given any two nodes $u, v \in V$, let $d(u, v)$ be the length of the shortest path between them.

The MTSSR studied in Section III-B is a special case of the Q-MTSSR wherein $pr_i = cost_i = 1$ and $\mathcal{Q} = q$. Therefore Q-MTSSR is at least NP-hard. Below we first show that DAO^2 in sensor network G is equivalent to Q-MTSSR in the aggregation network G' defined in Section III-C. We design approximation and heuristic algorithms on G' for Q-MTSSR and illustrate them using the DAO^2 example in Fig. II.

Theorem 4: DAO^2 in G is equivalent to Q-MTSSR in G'.

Proof: In addition to the proof for Theorem 3, which shows that all related information for energy-efficient data aggregation in sensor network $G(V, E)$ (i.e., data, energy cost, and topology) are all preserved in aggregation network $G'(V', E')$, we need to show that any instance of aggregating overflow data in DAO^2 is equivalent to an instance of collecting prizes in Q-MTSSR.

First, in DAO^2, when data node i is visited by an initiator, the amount of data it can reduce at i is $R_i \cdot \rho$. This is the amount of prize pr_i in Q-MTSSR that a traveling salesman can collect when he visits i. Second,

[1] $\sum_{k \in V'} pr_k > \mathcal{Q}$; otherwise the problem is not feasible.

[2] This models the general scenario wherein salesmen have different traveling means (i.e., cars or planes) thus incurring different costs per unit distance.

in DAO^2, the total amount of data sizes in G that needs to be reduced is $\mathcal{Q} = \sum_{i \in V_d} R_i - \sum_{j \in V - V_d} m_j$. In Q-MTSSR, this is indeed the total amount of collected prizes $\mathcal{Q}$ when selected traveling salesmen visit other cities. Third, the aggregation costs in DAO^2 depend on the sizes of the data packets transmitted from initiators, which corresponds in Q-MTSSR that salesmen dispatched from different cities have different traveling costs per unit distance. As any Q-MTSSR instance corresponds to a DAO^2 instance, solving Q-MTSSR in G' is equivalent to solving DAO^2 in G. ∎

B. Approximation Algorithms for Q-MTSSR

Definition 6: (**Prize of a Component, Distance between Two Components, Prize-Cost Ratio of Two Components**) The *prize of a connected component* C_i in G', denoted as $pr(C_i)$, is the sum of prizes on all nodes in C_i; i.e., $pr(C_i) = \sum_{u \in C_i} pr_u$. Given any two connected components C_i and C_j in G, their *distance*, denoted as $d(C_i, C_j)$, is the smallest length of all the shortest paths between two nodes, one is in C_i and the other in C_j; i.e., $d(C_i, C_j) = \min\{d(u, v) | u \in C_i, v \in C_j\}$. The *prize-cost ratio*, denoted as $pcr(C_i, C_j)$, is the ratio of the smaller prize of C_i and C_j to the distance between them; i.e., $pcr(C_i, C_j) = \frac{\min(pr(C_i), pr(C_j))}{d(C_i, C_j)}$. □

The rationale of Algo. 2 below is to utilize pcr to collect as much prizes as possible while using as least costs as possible. It starts with n components, each of which is one node and each is its own traveling salesman (line 1-2). In each round, it joins two components with the largest pcr using the shortest path between them (ties are broken randomly), takes the smaller ID of these two components as the ID of the new component, and collects their prizes as well as the prizes on the nodes along the connecting shortest path (line 3-11). In this newly combined component, the node with smallest prize is updated as the starting node (i.e., the node to dispatch the traveling salesman) (line 12-15). This continues until the total prize of all the connected components (i.e., which are trees) reaches $\mathcal{Q}$. Finally, a traveling salesman is dispatched from the starting node in each component to visit all other nodes to collect prizes by traversing each edge at most twice, with total collected prizes and total cost returned (line 17-22). The time complexity of Algo. 2 is $O(\mathcal{Q} \cdot |V'|)$.

There are two reasons we select the node with smallest prize in each component to dispatch the traveling salesman. First, by doing so, other nodes' larger prizes can thus be collected, which expedites collecting the target quota. Second, in DAO^2, as data packets from initiators are used to perform data aggregation on other nodes, data nodes with smallest packet sizes thus should be selected in order to save energy. This corresponds to that in Q-MTSSR node with smallest prize in each component dispatches the traveling salesman.

Algorithm 2: Approximation Algorithm for Q-MTSSR.

Input: $G(V', E')$, pr_u at node u, targeted quota $\mathcal{Q}$;

Output: total collected prizes $quota$, total traveling cost $\mathcal{T}$;

0. **Notations**:
   $quota$: prizes collected so far, initially zero;
   $\mathcal{C}$: the set of all the connected components,
   $\mathcal{C} = \{C_1, C_2, ..., C_{|V|}\}$, initially $C_i = \{i\}$, $\forall i \in V'$;
   A: IDs of all the resultant components, initially
   $A = \{1, 2, ..., |V'|\}$;
   ts_i: the starting node in C_i, initially node i;
1. $quota = 0$;
2. $ts_i = i$, $\forall i \in V'$;
3. **while** ($quota \leq \mathcal{Q}$)
4. Let $(C_{i^*}, C_{j^*}) = \text{argmax}_{(i,j),\text{where } i,j \in A, i \neq j} pcr(C_i, C_j)$;
5. Let $d(u, v) = d(C_{i^*}, C_{j^*})$, where $u \in C_{i^*} \wedge v \in C_{j^*}$;
6. Let $E(u, v)$ be all edges on the shortest path btw u, v;
7. Let $N(u, v)$ be all the nodes on the shortest path
   between and excluding u, v;
8. $a = \min\{i^*, j^*\}$ and $b = \max\{i^*, j^*\}$;
9. $x = \text{argmin}_{u \in C_a} pr_u$; // starting node before merge
   // Next merge C_b and $E(u, v)$ into C_a, take C_a's ID;
10. $C_a = C_a \cup C_b \cup E(u, v)$; $A = A - \{b\} - N(u, v)$;
11. $quota+ = \big(pr(C_{i^*}) + pr(C_{j^*}) + \sum_{i \in N(u,v)} pr_i\big)$;
12. $y = \text{argmin}_{u \in C_a} pr_u$; // starting node after merge
13. **if** ($pr_y < pr_x$) // update starting node in C_a
14. $ts_a = y$; $quota = quota - pr_y + pr_x$;
15. **end if;**
16. **end while;**
18. **for** (each element z in A)
19. **if** ($|C_z| \geq 2$) // has at least two nodes
    Dispatch a salesman from ts_z, who visits each node
    in C_z by traversing each edge at most twice, let
    the resultant walk be W_z;
    $\mathcal{T}+ = c(ts_z, W_z)$;
20. **end if;**
21. **end for;**
22. **RETURN** $quota$ and $\mathcal{T}$.

EXAMPLE 4: Fig. 7(a) shows the aggregation network for the DAO^2 example in Fig. II, with $\mathcal{Q} = 2$

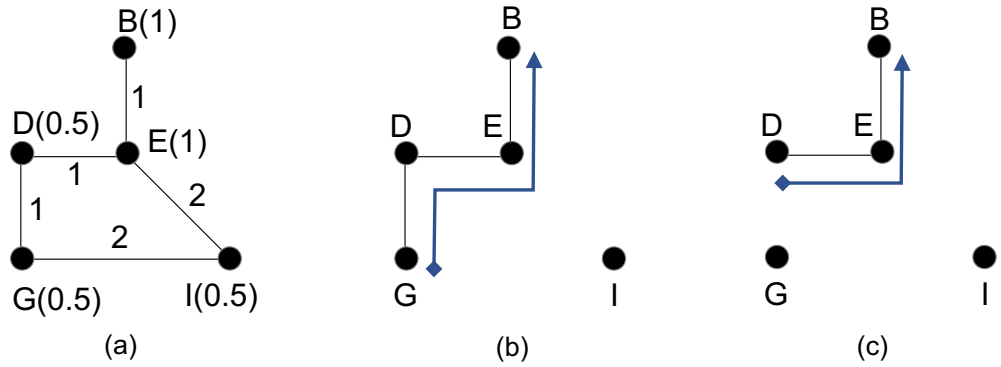


Fig. 7. (a) Aggregation network G' of sensor network G in Fig. II. The numbers in the parentheses are the prizes available at each data node. (b) and (c) are two of many possible solutions, with (c) being the optimal. Algo. 2 outputs both (b) and (c) whereas Algo. 3 outputs only (c).

and the prizes (i.e., data reduction) at each data node is $pr_B = pr_E = 1$ while $pr_D = pr_G = pr_I = 0.5$. Fig. 7(b) and (c) show two solutions by Algo. 2. In Fig. 7(b), G dispatches a salesman to visit D, E and B, collecting total prizes of 2.5 with total traveling cost of 3. Fig. 7(c) indeed gives the optimal solution wherein D dispatches a salesman to visit E and B, collecting total prizes of 2 with total traveling cost of 2. □

Algo. 2 is inspired by Awerbuch et al. [8], which solves a special case of Q-MTSSR. In particular, they considered a *quota-driven salesman probelm* in which a single traveling salesman collects prizes at different cities in order to reach a target quota while minimizing the traveling cost to reach the quota. It proposed an $O(\log^2 R)$ approximation algorithm where R is the quota. It is based on an approximation for the k-minimum-spanning-tree problem (k-MST), which is finding a tree of least weight that spans exactly k vertices on a graph. We thus give below theorem without proof.

Theorem 5: When only one traveling salesman is allowed, Algo. 2 achieves $O(\log^2 R)$ approximation for Q-MTSSR.

C. Heuristic Algorithm for Q-MTSSR

Algo. 2 iteratively combines two components that yield the maximum prize-cost ratio in each round until prize quota $\mathcal{Q}$ is reached. One drawback of this method is that if two components are distant from each other, connecting them could yield prizes that are much larger than $\mathcal{Q}$ (line 10 and 11, Algo. 2) thus cost more energy than necessary. For example, Fig. 7(b) shows a solution with collected prizes of 2.5, 25% more than target quota $\mathcal{Q} = 2$, and cost of 3, 50% more than the optimal cost of 2 obtained in Fig. 7(c). We thus design a cost efficient prize-collecting scheme that takes place on a local and more granular level than Algo. 2, and show that it constantly outperforms Algo. 2. We first give below definition.

Definition 7: (**Benefit of An Edge**) The *benefit of edge* $e = (u, v)$, denoted as $\mathcal{B}(e)$, is the ratio of the sum of the prizes of its two end nodes to its weight; $\mathcal{B}(e) = \frac{pr_u + pr_v}{w(e)}$. □

Algo. 3 below iteratively adds cycleless edges with maximum benefit until $\mathcal{Q}$ or slightly higher amount of prizes is collected. There are three possible cases when adding a cycleless edge e_i: a) it initiates a new connected component (line 6-9), or b) it connects two existing components (line 10-14), or c) it merges into one existing component (line 15-24). In each case, the starting node and the collected prizes are updated accordingly. Finally, the traveling salesman is dispatched in each component to visit all other nodes to collect prizes by traversing each edge at most twice, with total collected prizes and total cost returned (line 28-30). The time complexity of Algo. 3 is $O(|E'|\log|E'| + \mathcal{Q})$.

Algorithm 3: Heuristic Algorithms for Q-MTSSR.

Input: $G'(V', E')$, prize pr_u at node u, targeted quota $\mathcal{Q}$;

Output: total collected prizes $quota$, total traveling cost $\mathcal{T}$;

0. **Notations**:

   E_{pc}: set of cycleless edges selected for quota-collecting;

   $quota$: prizes collected so far, initially zero;

   $sel(u)$: if node $u \in V$ is selected, initially false;

   i: indices for edges; j: indices for components;

   a: number of connected components created;

   $\mathcal{C}$: the set of all the connected components,

   $\mathcal{C} = \{C_1, C_2, ..., C_a\}$, initially $C_i = \phi$;

   A: IDs of the final connected components;

   ts_j: the traveling salesman in component C_j, $1 \leq j \leq a$;

1. $\mathcal{B}(e_1) \geq \mathcal{B}(e_2) \geq ... \geq \mathcal{B}(e_{|E|})$; // Sort edges in $\mathcal{B}$

2. $i = 1$, $quota = 0$, $a = 0$, $A = E_{pc} = \phi$ (empty set);

3. **while** ($quota \leq \mathcal{Q}$)

4. **if** (e_i causes a cycle w.r.t. E_{pc}) continue;

5. Let e_i's two end nodes be n_1 and n_2;

   // e_i initiates a new component

6. **if** $(sel(n_1) == sel(n_2) == false)$

7. $a++$, $A = A \cup \{a\}$;

8. **if** $(pr_{n_1} \leq pr_{n_2})$ $ts_a = n_1$, $prize+ = pr_{n_2}$;

9. **else** $ts_a = n_2$; $prize+ = pr_{n_1}$;

   // e_i connects two existing components

10. **elseif** $(sel(n_1) == sel(n_2) == true)$

Let the two components are C_b and C_c and $b \leq c$;
$y_b = \text{argmin}_{u \in C_b} pr_u$, $y_c = \text{argmin}_{u \in C_c} pr_u$;
$ts_b = \min\{y_b, y_c\}$, $ts_c = \max\{y_b, y_c\}$;
// Merge C_c into C_b
$C_b = C_b \cup C_c$, $A = A - \{c\}$, $quota+ = pr_{ts_c}$;
else // e_i merges into one existing component
Let the component e_i merges into is C_b;
$y = \text{argmin}_{u \in C_b} pr_u$;
if $(sel(n_1) == false)$ $x = n_1$;
else $x = n_2$; // $(sel(n_2) == false)$
$sel(x) = true$;
if ($pr_x < pr_y$) // update starting node in C_b
$ts_b = x$; $quota+ = pr_y$;
else $quota+ = pr_x$;
end else
$E_{pc} = E_{pc} \cup \{e_i\}$;
$i++$;
end while;
for (each element z in A)
Dispatch a salesman from ts_z, who visits each node in C_z by traversing each edge at most twice, let the resultant walk be W_z;
$\mathcal{T} += c(ts_z, W_z)$;
end for;
RETURN $quota$ and $\mathcal{T}$.

EXAMPLE 5: In Fig. 7(a), as $\mathcal{B}(B,E) \geq \mathcal{B}(D,E) \geq \mathcal{B}(D,G) \geq \mathcal{B}(E,I) \geq \mathcal{B}(G,I)$, Algo. 3 selects edges (B,E) and (D,E) and dispatches salesman from D to visit E and B, which is the optimal solution shown in Fig. 7(c). □

V. Distributed Data Aggregation Algorithms

We design a distributed algorithm, referred to as *Distributed DAO*^{2}, to solve aforesaid data aggregation problem. It consists of three stages. First, it constructs the aggregation network of the data nodes $G'(V', E')$ from the sensor network $G(V, E)$ by modifying the distributed Bellman-Ford algorithm [37]. Second, the data nodes in the aggregation network cooperatively find the q-edge forest based on a classic distributed MST algorithm [23, 40]. Third, of each tree in the q-edge forest, an initiator is selected and

starts the data aggregation process to reduce the overflow data size. Below we illustrate each stage in details.

As the energy consumption is a critical factor for measuring the efficiency of distributed algorithms in wireless ad hoc networks, there are also work that strives to achieve energy-efficient distributed MST construction. For example, Choi [16] designed energy-optimal distributed MST algorithm with $O(|V|\log|V|)$-approximation instead of optimal. We takes into account energy consumption in our design but still strives to the minimum q-edge forest as in our centralized algorithms.

Constructing Aggregation Network. The distributed Bellman-Ford (DBF) algorithm, also called distance vector protocol in network community, is a well-established asynchronous technique to compute shortest paths between nodes in a network distributedly. Following DBF, each node (data node or storage node) initially only has direct knowledge of its local links but sends message about its perceived shortest path lengths to all other nodes (i.e., the routing table) to its neighbors. When a neighbor receives the message, if it finds that its current cost to a node is greater than the sum of its cost to the sender and the sender's cost to that node, it update its routing table and send it to its neighbors. This takes place iteratively and asynchronously until all the nodes have the accurate shortest paths information to other nodes in the network. The message size in DBF is $O(|V|)$, where $|V|$ is number of nodes.

However, there are two challenges to apply DBF directly to construct aggregation network. First, it is well known that message complexity of asynchronous DBF could be exponential [9]. Second, and on top of DBF, the aggregation network is constructed by checking if the shortest path between two data nodes contain another data node. To overcome these challenges, we have made two improvement upon existing DBF. First, unlike classic DBF wherein the messages are unicast, the wireless communication is generally broadcast where all nodes within the transmission range of the sender receives the message. Using a combination of unicast and broadcast, the message complexity of DBF is reduced to $|V| \cdot |V|$. Second, in order for data nodes to find out if it has a "direct" link with another data node in the aggregation network, the message sent by each node (both data node and storage node) includes its perceived shortest path length to other nodes as well as the shortest path itself. The message size becomes $O(|V|^2)$. With this information, any data node checks its shortest path to all other data nodes; if there is no other data node on the shortest path, it has an edge to the other data node in the aggregation network and the cost of the edge is the cost of the corresponding shortest path in the sensor network. At the end of this stage, the aggregation network is constructed as any data node knows not only which data node it has an edge with but also the cost of this edge in the aggregation network.

Constructing q-edge Forest. Next, the p data nodes in the constructed aggregation network $G'(V', E')$ cooperate to find a q-edge forest among them in a distributed manner. We propose two implementations. Naive Distributed Algorithm. The naive approach serves as the baseline algorithm to be compared with the other distributed algorithm discussed below. As at the end of aggregation network construction, each

data node not only knows the IDs of other data nodes but also the shortest paths to them, the node with smallest ID is thus selected as the leader. Then every node just sends the weights of all its incident edges to the leader following the shortest path between it and the leader. Once the leader receives such information from all the nodes, it executes Algo. 1 to find the minimum q-edge forest of the aggregation network. Finally, it broadcasts this result to all the data nodes of the aggregation network. When each data node receives it, it checks if it is an end node of any of computed edges; if so, it marks these edges as *tree edges*.

GHS-based Distributed Algorithm. Our second approach is based on the classic GHS algorithm, a distributed and asynchronous MST algorithm [23, 40]. The main idea is to maintain a forest of spanning trees, each is called a fragment, until q edges are included. Initially each node is a fragment with level 0 and the node ID being the fragment's ID. It then repeatedly merges fragments until q edges are found. However, as an MWOE found does not necessarily belong to the minimum q-edge forest, the resulted q-edge forest could be sub-optimal (we compare it with the above naive approach in terms of solution quality and energy consumptions in Section VI). Each fragment continuously and independently executes below two stages viz. finding the MWOE and combining with other fragments via the MWOE.

1. Finding minimum weight outgoing edge (MWOE). MWOE is an edge of minimum weight with its two endpoints on two different fragments. Each fragment finds its MOWE independently and uses it to combine with other fragments. In the process each edge must be one of the three below: *tree edges* that have been determined as edges in the q-edge forest, *rejected edges* that have been determined not, and *basic edges* that have not been decided. The MOWEs are indeed tree edges. Initially each node (level-0 fragment) marks its minimum-weight edge as a tree edge and sends a message to the node on the other side. The edge chosen by both nodes then combines these two nodes, which becomes a new fragment with level 1.

For each non-level-0 fragment to find its MWOE, its leader, which is one of the end nodes with smaller ID of the MWOE added previously, sends an *initiate* message to the members of the fragment along the *tree edges*. Upon receipt of the message, each node n sends its fragment ID and level along its *basic edges* to node n' on the other side. Then n' compares them with its own fragment ID and level, and make the following decisions. (a) If FragmentID(n) = FragmentID(n'), then n and n' belong to same fragment thus they mark the edge as a *rejected edge*; (b) if FragmentID(n) != FragmentID(n') $\wedge$ Level(n) $\leq$ Level(n'), then n and n' belong to different fragments thus n' sends a message to n about this outgoing edge; (c) if FragmentID(n) != FragmentID(n') $\wedge$ Level(n) $>$ Level(n'), n' postpone the response until Level(n') $\geq$ Level(n).

Next, all the leaves in the fragment sends its observed MWOE (if there is any) along the tree edge back to its parent in the fragment. For each non-leaf node, after receiving all the MWOE messages, it finds the MWOE with minimum weight and sends it to its parent. This takes place until the leader

receives the MWOE messages from all its neighbors and identifies the MWOE for the entire fragment. Finally the leader sends a broadcast message about this new MWOE to the entire fragment via the tree edges, and starts the fragment combining process as described below.

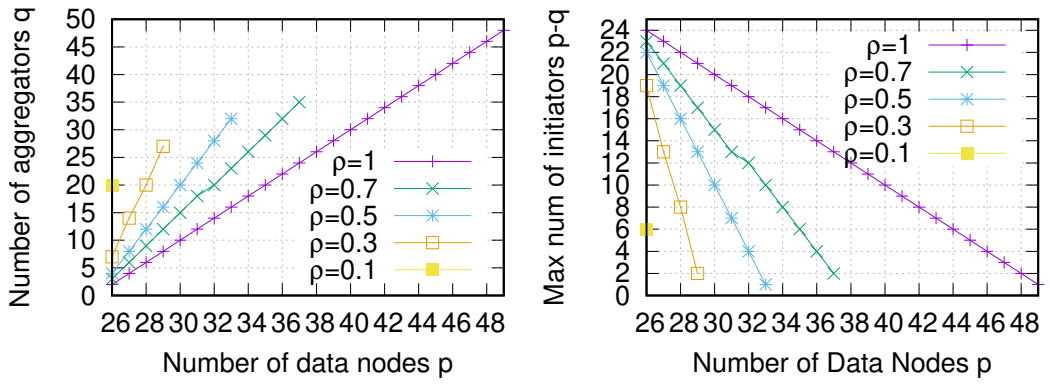
2. Combining fragments via their MWOEs. Upon receipt of such message, the end node of this MWOE that is within the fragment, say n, becomes the new leader. It marks this edge as a tree edge and sends a "request to combine" message to the other end of the MWOE, say n'. Under below two scenarios these two fragments will be combined. First, if n' selects the same edge as its MWOE and Level(n) = Level(n'), then the level of the combined fragment increases by one, and the end node of the MWOE with larger ID becomes the leader of this new fragment (and this ID becomes the ID of this newly formed fragment). This is called a "merge" operation. Second, if Level(n) $<$ Level(n'), then FragmentID(n') and Level(n') become the ID and level of the new fragment respectively. This is called "absorb" operation. For all other scenarios, n' ignores the "request to combine" message from n.

Finally, the leader in the combined fragment broadcasts a "new-fragment" message to the entire fragment edges and the MOE edges chosen by all the fragments. Upon receipt of it, all the nodes update their parent, children, and fragment identifier (which is the ID of the new leader). The distributed algorithm terminates when there are at least q found and included in the existing components.

In distributed MST algorithm, a MWOE must belong to the MST according to *cut property* of a graph, which says the minimum weight edge crossing any cut (i.e., fragment) in a graph must be in the MST. However, in our case of finding minimum q-edge forest, a MWOE found at a particular stage does not necessarily belong to the minimum q-edge forest. This is because in the distributed environment, a particular MWOE found in any stage could be an edge that is not one of the q smallest cycle-less edges. As such, we compare it with above Naive approach, which always finds the minimum q-edge forest.

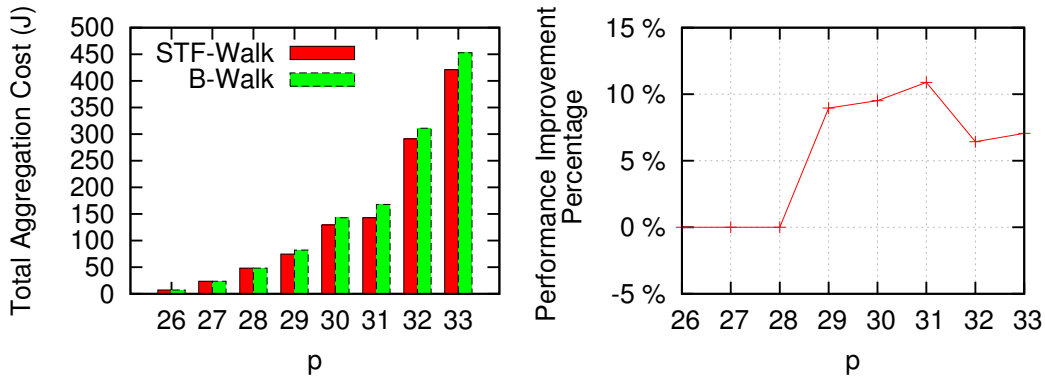
Data aggregations. In this final stage, an initiator in each connected component (i.e., tree) is selected, which then transmits its packet to visit all the nodes in the tree following the longest-path walk defined in III-B2. Thus the initiator is one of the two end leaf nodes of the longest path with the smaller ID. To find the initiator of a tree, we need to run the DBF algorithm again used in aggregation network construction stage in the tree to find the cost of the shortest (and only) path between any two leaf nodes. Then each leaf node broadcasts to the entire tree a message including its ID and the maximum path cost it has. After receiving all such messages, the leaf node with the maximum cost among all the maximum-cost paths (i.e., the longest path) and smaller ID knows it is the initiator. It transmits its data to visit all the nodes in the tree while traversing the edges on this longest-path once.

Time and Message Complexity. For the first stage of Distributed DAO^2, both its time and message complexities are $O(p|E|)$. For the second stage, its time complexity is $O(p \cdot \log p)$ while its message complexity is $O((p + |E|) \cdot \log p)$. Therefore, both the time and message complexities of Distributed DAO^2 are $O(p|E|)$.



(a) Number of aggregators q. (b) Max number of initiators $p - q$.

Fig. 8. Valid range of number of data nodes p by varying ρ ($R = m$).



(a) Total aggregation cost. (b) Performance improvement.

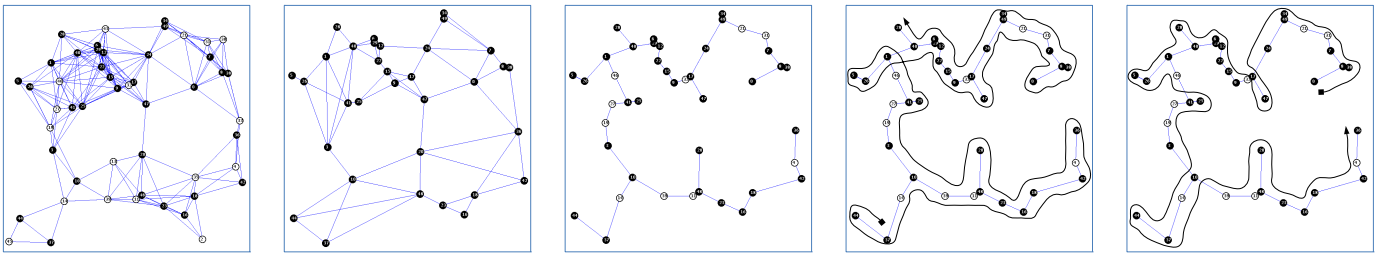
Fig. 9. Performance improvement of STF-Walk over B-Walk.

Theorem 6: When there is one initiator allowed, Distributed DAO^2 finds an optimal aggregation cost.

Proof: When there is only one initiator, finding minimum q-edge forest in data aggregation is equivalent to finding a MST in the aggregation network. Consequently, Distributed DAO^2 is equivalent to GHS algorithm. Due to the optimality of finding MST by the GHS algorithm, the optimality of Distributed DAO^2 also sustains. ∎

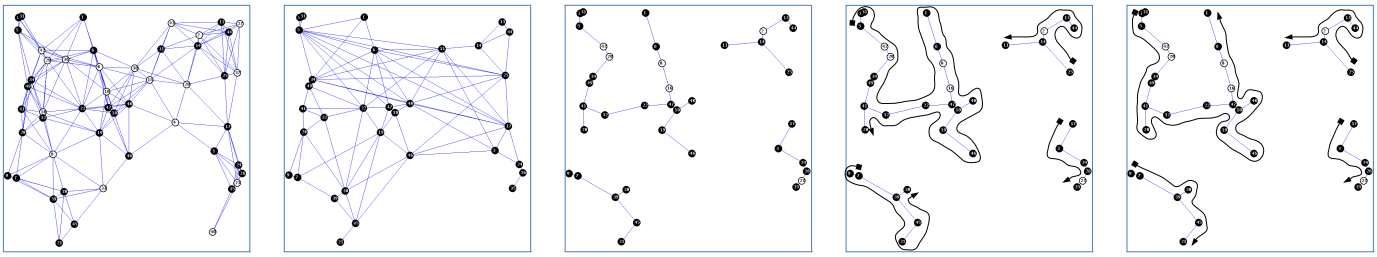
VI. Performance Evaluation

We implemented a Java-based simulator in which 50 and 100 sensors are uniformly distributed in a region of 1000m $\times$ 1000m or 2000m $\times$ 2000m. Transmission ranges of sensor nodes are set as 250m. Each data point is an average over 10 runs and the error bars indicate 95% confidence interval, wherever applicable. Below we evaluate the algorithms for DAO^2-U and DAO^2, respectively.



(a) Sensor network of 50 nodes. (b) Aggregation network. (c) 32-edge forest. (d) B-Walk (381.2KJ). (e) LP-Walk (290.6KJ).

Fig. 10. Visually comparing B-Walk with LP-Walk with one initiator. Black nodes are data nodes and white nodes are storage nodes, with node ID shown inside. Here, $\rho = 0.5$, $p = 33$, and $q = 32$. ■ and ◄– indicate the first and last node in a walk, respectively.



(a) Sensor network of 50 nodes. (b) Aggregation network. (c) 28-edge forest. (d) B-Walk (cost 255.9J). (e) LP-Walk (cost 203.0J).

Fig. 11. Visually comparing B-Walk with LP-Walk with 4 initiators. Black nodes are data nodes and white nodes are storage nodes, with node ID shown inside. Here, $\rho = 0.5$, $p = 32$, and $q = 28$. ■ and ◄– indicate the first and last node in a walk, respectively.

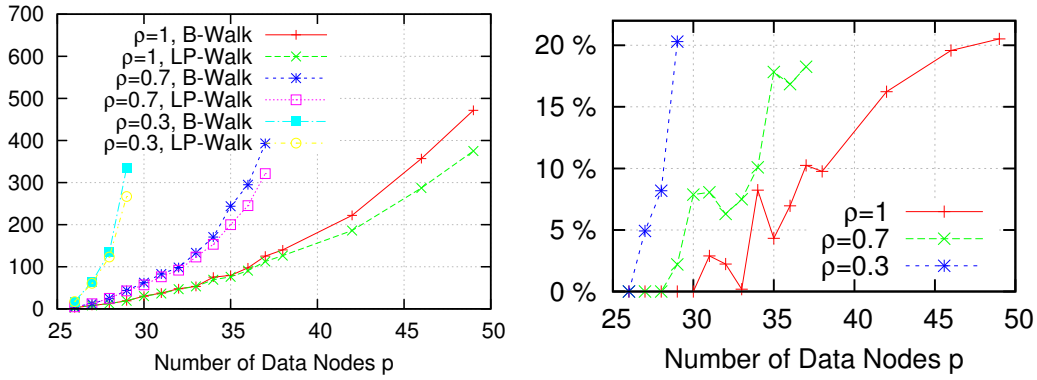
A. Algorithms for DAO^2-U.

1) Centralized Algorithms: We consider 50 nodes in a $1000\text{m} \times 1000\text{m}$ region, as they can be clearly visualized. Unless otherwise mentioned, $R = m = 512\text{MB}$.

Valid Range of p. Fig. 8 shows the valid range of number of data nodes p for different correlation coefficient ρ. Fig. 8(a) shows for each valid p value its corresponding value of number of aggregators q. When $\rho = 0.1$, the valid range of p is a single value of 26, with its corresponding q as 20. When increasing ρ, the valid range of p expands, from $26 - 29$ for $\rho = 0.3$, to $26 - 33$ for $\rho = 0.5$, to $26 - 37$ for $\rho = 0.7$, to $26 - 49$ for $\rho = 1$. This is because strong data correlation leads to more data being aggregated, thus allowing more data nodes under overall storage overflow. It also shows that for each ρ, q increases when increasing p. This is because more data nodes means more overflow data and less available storage, therefore more aggregators are needed to achieve enough data size reduction. Finally it shows that for the same p, q decreases when increasing ρ. This is implied by Equation 1, which can be rewritten as: $q = \lceil \frac{p \times (1+m/R) - |V| \times m/R}{\rho} \rceil$. Fig. 8(b) shows the maximum number of allowable initiators

$p-q$ for each valid p value. There are two cases in which one initiator is allowed: $\rho = 0.5$ and $p = 33$, and $\rho = 1$ and $p = 49$, while multiple initiators are allowed for other cases.

Performance Improvement of STF-Walk Over B-Walk. We first study the performance improvement of STF-Walk over B-Walk. We choose $\rho = 0.5$, which is a representative correlation coefficient, and vary p from 26 to 33. Fig. 9(a) shows that when p is 26, 27, or 28, both STF-Walk and B-Walk yield the same total aggregation costs. This is because when the number of data nodes p is small, the number of aggregators q is small, causing that the connected components of the resulted q-edge forests are all linear. In linear topologies, aggregation takes place by simply traversing from one end of the linear topology to the other end, resulting the same performances for both STF- and B-Walk. However, when p gets larger, STF-Walk yields less cost and performs better than B-Walk does, because STF-Walk always traverses the smaller subtree twice while B-Walk could possibly traverse the bigger subtree twice. Fig. 9(b) shows that the performance improvement of STF-Walk over B-Walk is around $5\% - 10\%$. Therefore, for the rest of the simulations we choose STF-Walk instead of B-Walk, but still refer to it as B-Walk.



(a) Total aggregation cost (KJ).

(b) Performance improvement.

Fig. 12. Comparing B-Walk with LP-Walk by varying p and ρ.

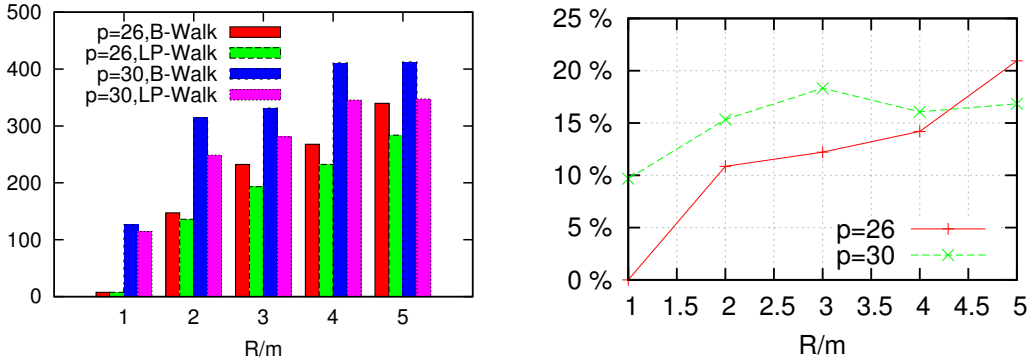
Comparing B-Walk with LP-Walk Visually. Before we perform a comprehensive comparison between B-Walk and LP-Walk, we first compare them visually to gain some insights.

Single Initiator Case. We consider $\rho = 0.5$ and $p = 33$, which has 32 aggregators and one initiator. Fig. 10(a) and (b) show such a sensor network and its corresponding aggregation network, respectively. Fig. 10(c) shows the corresponding 32-edge forest. Fig. 10(d) and (e) show the aggregation walks from B-Walk and LP-Walk, respectively. B-Walk visits 32 edges twice, resulting in a total aggregation cost of 381.2 kilojoules (KJ); while LP-Walk only visits 12 edges twice, with a total cost of 290.6KJ, a 23.8% of improvement upon B-Walk. Here, each edge in the 32-edge forest is replaced by a shortest path of storage nodes, if there are any. Therefore there are more than 32 edges in this forest.

Multiple Initiators Case. We consider $\rho = 0.5$ and $p = 32$, which has 28 aggregators and allows at most 4 initiators. Fig. 11(a) and (b) show such a sensor network and its corresponding aggregation network.

Fig. 11(c) shows the 28-edge forest of the aggregation network. It consists of four "clusters", each of which is visited by one initiator. However, Fig. 11(d) and (e) show that how B-Walk and LP-Walk visit each cluster differently. It shows that B-Walk traverses 19 edges twice, resulting in a total aggregation cost of 255.9J, while LP-Walk traverses 9 edges twice, with a total aggregation cost of 203.0J, a 20.7% of improvement. Compared to Fig. 10, this shows that when increasing number of initiators, the performance difference between B-Walk and LP-Walk gets smaller. In particular, Fig. 11(d) and (e) show that they find exactly the same aggregation walks for two smaller trees on the right. With more initiators allowed, the resulted q-edge forest consists of more trees with small sizes, each with a "short" longest path. By traversing edges on such short longest paths once, LP-Walk does not save as much at it does compared to traversing a big tree with much longer longest path. Finally, compared to single initiator case, both B-Walk and LP-Walk incur less energy cost, as more initiators are utilized to find cost-effective aggregation walks.

Comparing B-Walk with LP-Walk. Next we compare the aggregation costs in B-Walk and LP-Walk in the whole range of $p \in [26, 49]$ with varied ρ. Fig. 12(a) shows that for each ρ, with the increase of p, the total aggregation costs of both B-Walk and LP-Walk increase. However, LP-Walk constantly performs better than B-Walk. It also shows that for the same p, with the increase of ρ, the aggregation costs for both B-Walk and LP-Walk decrease. This is because more correlation means that less number of aggregators are visited, thus incurring less aggregation costs.



(a) Total aggregation cost (KJ).

(b) Performance improvement.

Fig. 13. Comparing B-Walk with LP-Walk by varying R/m.

Fig. 12(b) shows the performance improvement percentage of LP-Walk over B-Walk is generally $10\%-20\%$. Combining the $5\% - 10\%$ performance improvement of STF-Walk over B-Walk, the performance improvement of LP-Walk over B-Walk is therefore around $15\% - 30\%$. Furthermore, we observe the smaller the ρ, the larger the performance improvement percentage is. For example, when $p = 26$ (the only valid value for $\rho = 0.1$), the performance improvement percentage for $\rho = 0.1$ is 14% while zero for $\rho = 0.3, 0.5, 0.7, 1.0$. When $\rho = 0.5$, in its valid p range $(26 - 33)$, it almost always has a larger

performance improvement percentage compared to $\rho = 0.7, 1$. When less data correlation exists, more aggregators are visited, making the sizes of the resulted q-edge forest as well as its constituent trees larger. By traversing the longest paths of larger trees once, LP-Walk can thus save more aggregation cost compared to traversing smaller trees.

Comparing B-Walk with LP-Walk by Varying R/m. We compare B-Walk with LP-Walk on different R/m. When increasing R/m, the overall storage overflow situation gets more challenging since there are relatively more overflow data compared to available storage spaces. We choose $\rho = 0.5$ and vary R/m from 1 to 5, under which the common valid range of p is $[26, 30]$. Therefore we pick $p = 26$ and $p = 30$ for comparison. Fig. 13(a) shows again that LP-Walk yields less total aggregation cost under different R/m. Fig. 13(b) further shows that the performance improvement percentage of LP-Walk upon B-Walk generally increases when increasing R/m. This shows LP-Walk performs even better in more challenging overall storage overflow scenarios. When increasing R/m, the resulted q-edge forests get larger. This favors LP-Walk, which travels large amount of edges only once.

2) Distributed Algorithm: Next we evaluate the performances of our designed distributed algorithms. We write our own simulators in Java on MacBook Pro with 2 GHz Dual-Core Intel Core i5 processor and 8 GB RAM. 100 nodes are randomly placed in a $2000\text{m} \times 2000\text{m}$ region. The transmission range is 250m and $m = R = 512$MB. Fig. 14 shows different kinds of messages (overhead and data) in our distributed algorithms with their sizes. The overhead messages are used in aggregation network construction using distributed Bellman-Ford algorithm, constructing the q-edge forest, as well as the data aggregation stage to choose an initiator for each tree. The data messages are used in data aggregation stage by that initiators send their overflow data packets to visit other aggregators for aggregations. As the overflow data packet size of 512 MB is very large, it can be fragmented in many smaller data packets.

	Constructing Aggregation Graph	Naïve Approach		GHS Approach	Aggregation Walk
		Nodes update their incidents edges to Leader	Leader informs computing result		
Message Size	1000 Bytes	20 Bytes	1000 Bytes	20 Bytes	512 Mbytes

Fig. 14. Sizes of different messages.

To compare different algorithms, we adopt a typical correlation coefficient value $\rho = 0.6$ and vary p from 51 to 71, at which point it allows for one initiator. We vary number of data nodes p in its valid range [55,71]. Fig. 15 compares the total energy costs of the two distributed algorithms viz. Naive and GHS-Based. We observe that both distributed algorithm perform close in terms of total energy cost while Naive performs slightly better.

Fig. 16 further shows the number of initiators and breakup of the energy cost in both distributed

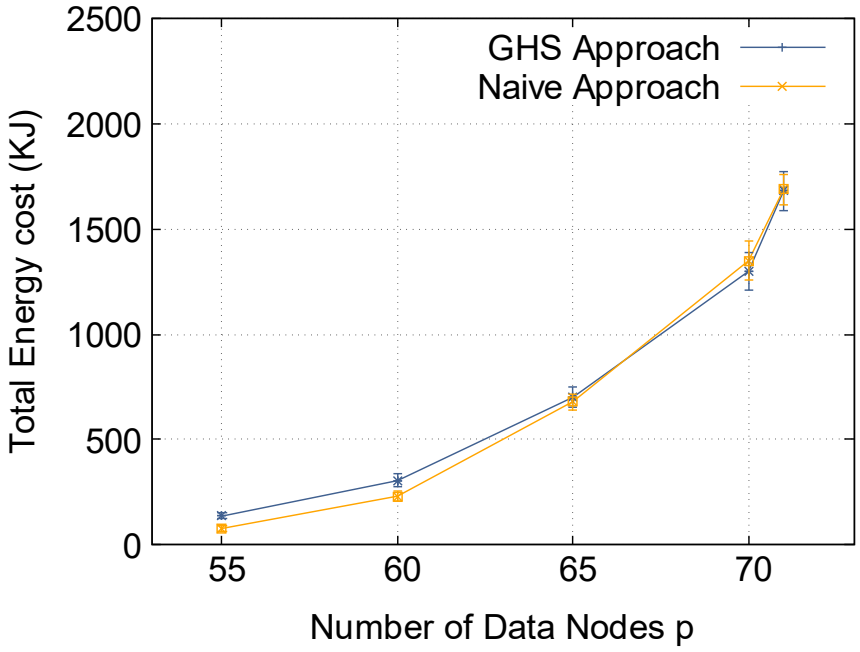


Fig. 15. Total energy costs in Naive and and GHS-Based distributed algorithms.

algorithms. It is interesting to observe that with the increase of p, the number of data nodes, the number of initiators needed for energy efficient data aggregation in both algorithms increases first and then decreases. During the initial stage of increasing p with increased amount of overflow data, the more number of aggregators need to be visited in order to achieve the required amount of data reduction. With the increasing number of aggregators, more connected components (i.e., trees with at least two nodes) are formed. As each tree needs one initiator, this results in increased number of initiators in the initial stage of increasing p. However, with the further increase of p, the number of aggregators increases to an extent that the different trees they belong to begin to merge, resulting in less number of trees thus less number of initiators. As the maximum allowable number of initiators decreases with the increase of p fig:feasible1, the number of initiators finally gets to 1.

Finally, we observe that among the three stages in the distributed algorithms, finding q-edge forest costs minimum portion of energy, ranging from 1.8% in the smaller number of p to 0.3% at the larger number of p. Then it is constructing aggregation networks, with energy cost ranging from 38.5% in the smaller number of p to 3.2% at the larger number of p. This shows that in a really data-intensive intermittently connected sensor networks., the majority of the sensor energy consumption is spent on data aggregation payload, instead of on the overhead cost spent in aggregation network construction and finding q-edge forest. This demonstrates the energy-efficiency of our data preservation system.

Number of Initiators, edges

p	55	60	65	70	71
q	17	34	50	67	70
Number of Initiators					
Naïve Approach	11.3	13.2	9.5	2.5	1
GHS Approach	14.8	18.9	14.3	2.8	1
Number of edges					
Naïve Approach	17	34	50	67	70
GHS Approach	17	34	50	67	70

Naïve Approach

p	55	60	65	70	71
Constructing Aggregation Graph (J)	55.26	51.42	53.23	54.27	55.4
Leader-Nodes communicaton (J)	2.85	3	3.24	3.32	3.42
Aggregation Cost (KJ)	77.79	228.62	684.42	1327.36	1682.95

GHS Approach

p	55	60	65	70	71
Constructing	55.26	51.42	53.23	54.27	55.4
GHS (J)	0.81	1.74	2.85	4.2	4.55
Aggregation Cost (KJ)	134.25	300.82	696.48	1271.86	1674.81

Fig. 16. Comparing Naive and GHS distributed algorithms.

B. Algorithms for DAO^2.

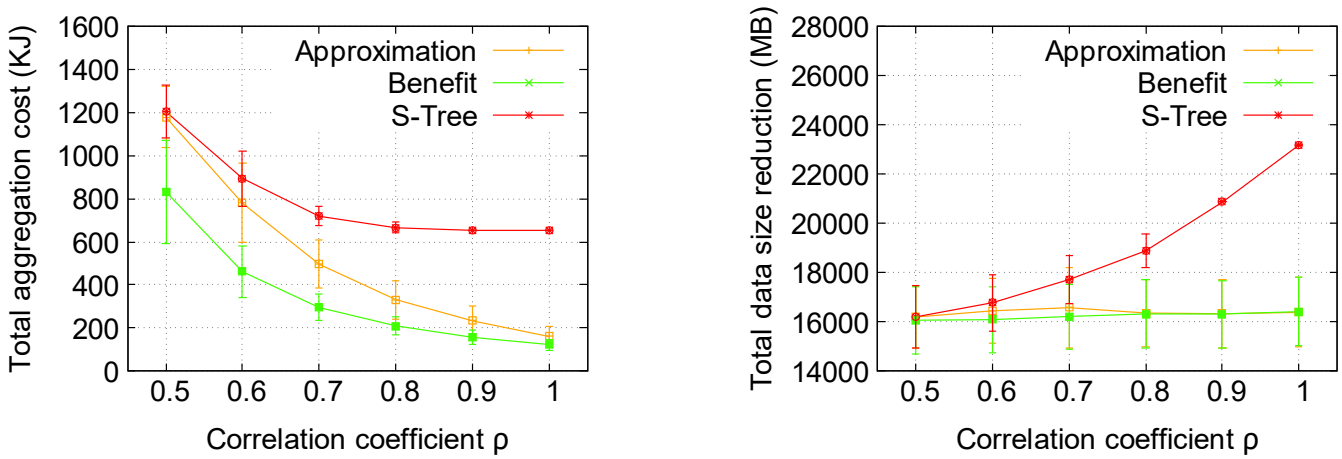
1) Centralized Algorithms: We randomly generate 100 sensor nodes (50 data nodes and 50 storage nodes) in a $2000\text{m} \times 2000\text{m}$ field with transmission range being 250m. In this network. R_i is a random number in [512MB, 1024MB] and m_j is a random number in [256MB, 512MB]. We first calculate the ρ_{th} defined in Section II, the smallest effective correlation coefficient that can be used to aggregate data. We then vary ρ by incrementing from ρ_{th} to 1 in 0.1 stepwise.

State-of-the-art Data Aggregation Techniques. Tree-based routing structures connecting base station and sensor nodes are often used in existing data aggregation techniques in sensor networks [4, 15, 29, 31, 34, 47]. Following such aggregation tree, data at sensor nodes are transmitted back to the base station while being aggregated along the way. As existing data aggregation works only consider a single aggregation tree due to including of the base station, we thus construct one aggregation tree in DAO^2 to compare with existing work. To make a fair comparison, though, as there is no base station in DAO^2 and its goal is to aggregate data locally on data nodes, we construct the corresponding single data aggregation tree by modifying Algo. 2. In particular, it merges two components with largest prize-cost ratio until its largest component collects enough quota of $\mathcal{Q}$. This largest component serves as the data aggregation tree for existing work. Then, the node with smallest prize in this tree is selected as the base station (i.e.. the initiator), from which its overflow data is transmitted to visit all the data nodes in the tree to aggregate

their overflow data by traversing each edge at most twice.

We refer to the approximation algorithm Algo. 2 as **Approximation**, the heuristic algorithm Algo. 3 as **Benefit**, and the existing aggregation technique with available base station as **S-Tree**, as it requires a single tree connecting the sensors and base station. Fig. 17 compare the three algorithms by varying the correlation coefficient ρ. Fig. 17(a) shows that in terms of the total aggregation cost, Benefit outperforms Approximation, which outperforms S-Tree. When ρ is small, as large number of aggregators are needed to reach the required data size reduction, it is likely that those aggregators are connected to form one or multiple large components. Visiting all the aggregators in these large components from a few initiators is more costly than visiting the aggregators from different initiators, which is more energy-efficiently. Therefore when ρ is small, the total costs of both Approximation and Benefit are close while both are larger than the cost of Benefit. When ρ gets larger, as less number of aggregators are needed, each components in Approximation gets smaller, resembling those in Benefit, thus the performance of Approximation gets close to Benefit. However, as S-Tree always requires one single connected component for aggregation, its cost stays much higher.

Fig. 17(b) shows the total size reduction achieved in all three algorithms(while the average required overflow data size $\mathcal{Q}$ is 15904 MB). It shows that both Approximation and Benefit reduces around 16 MB data size for the entire range of ρ, around 3% of more reduction than required. A close looks shows that Approximation aggregates more data than Benefit at $\rho = 0.6$ and 0.7, sustaining our initial conjecture that Approximation could yields more prizes (i.e., aggregates more data) than Benefit if the components to be connected are distant from each other. In contrast, S-Tree always look for a single aggregation tree to include all the targeted aggregators. With the increase of ρ, as the less number of aggregators are needed, S-Tree thus needs more "connecting nodes" to connect the targeted aggregators, thus increasing the total size of data reduction.



(a) Total aggregation cost (KJ).

(b) Total data size reduction.

Fig. 17. Performance comparison of the three algorithms.

Table II shows the number of initiators p number of aggregators q for the three algorithms. With the increase of ρ, q decreases for all three algorithms – as each data node can aggregate more data, it needs less number of data nodes to be aggregators. However, S-Tree has more aggregators than Approximation and Benefit at each fixed ρ value, showing that it is less cost efficient than the other two. We also observe that with the increase of ρ, number of initiators a increases for both Approximation and Benefit. This is because the less number of aggregators in the sensor field, the more distance among them thus the smaller chance that they are merged into components, resulting in more components. As each component has one initiator, thus the number of initiators increases with the increase of ρ. Note that as S-Tree always finds one aggregation tree, it has only one initiator.

ρ		Approximation	Benefit	S-Tree
0.5	a	1.5	6.7	1
	q	41.4	40.1	41.5
0.6	a	2.8	10.5	1
	q	34.3	32.3	35.7
0.7	a	4.5	11.2	1
	q	29.1	27.4	32.5
0.8	a	5.9	11.3	1
	q	24.5	23.8	30.5
0.9	a	6.6	10.2	1
	q	21.7	21.2	30
1	a	7.1	9.6	1
	q	19.6	19.2	30

TABLE II

INVESTIGATING NUMBER OF INITIATORS a AND NUMBER OF AGGREGATORS q W.R.T. CORRELATION COEFFICIENT ρ.

Fig. 18 visually shows the resultant components from a typical run of the three algorithms. Benefit has 13 components whereas Approximation has 4 and S-Tree has 1. This demonstrates the more granular effort of energy-efficient data aggregation in Benefit wherein each initiator only visits its local data nodes for aggregation. In contrast, in Approximation and S-Tree, initiators could travel long distance for aggregation thus costing more energy.

VII. **Related Work**

MSTW is different from well-known multiple traveling salesman problem (mTSP) [12] and vehicle routing problem (VRP) [48] studied in theory community. In mTSP, a group of traveling salesmen are given, and it needs to decide a tour for each salesman such that the total tour cost is minimized and that each city is visited exactly once. MSTW, however, needs to first decide how many salesmen can be dispatched (and from which cities), then to find the tour for each. Besides, not necessarily all the nodes

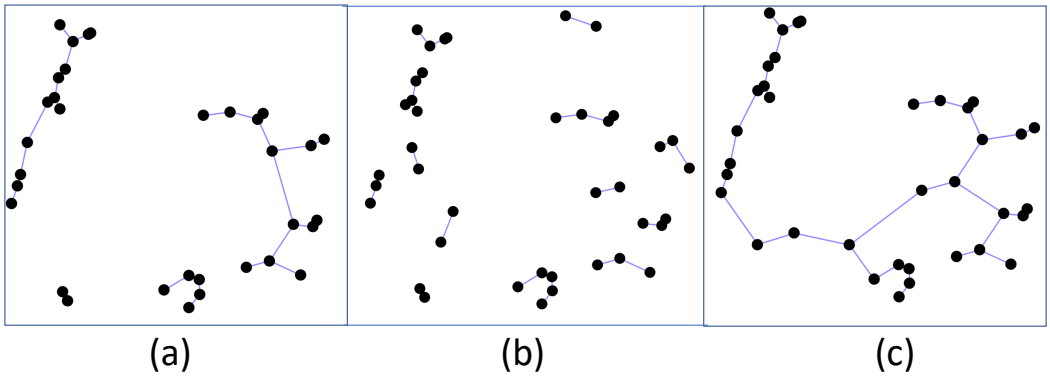


Fig. 18. (a) Approximation. (b) Benefit. (c) S-Tree.

will be visited in MSTW. In VRP, the set of vehicle nodes and the set of customer nodes are usually disjoint. There is no such distinction in MTSSR – each node can either dispatch a salesman or be visited.

The theoretical underpinning of DAO^2 is closely related to the *prize-collecting TSP* (or TSP/vehicle routing with profit) that has been studied extensively by theory community [6, 10, 21]. In prize-collecting TSP, each vertex has a prize (or profit) to be collected and a penalty if not visited; the goal of the traveling salesman is to minimize his travel costs and penalties while visiting enough cities to collect a prescribed amount of prize money. Note the complementary problem of maximizing the collected profit with the traveling cost not exceeding a budget is called orienteering problem or budgeted prize-collecting TSP [49]. As our problem is to aggregate and reduce the size of sensory data by at least some specified amount while minimizing the energy cost incurred, it is closer to the prize-collecting TSP and we review its related literature below.

Bienstock [13] studies a variation that finding a subset of vertices such that the length of the tour plus the sum of penalties associated with vertices not in the tour is as small as possible, and proposed a 2.5 approximation algorithm based on linear programming relaxation. Archer et al. [5] further improved the approximation ratio to $2 - \epsilon$ using Lagrangian relaxation. Tang and Wang [46] proposed a local search-based heuristic for a related capacitated prize-collecting TSP. Dell'Amico et al. [20] developed a lagrangian heuristic and obtained an upper bound in the form of a feasible solution. However, none of them considered multiple traveling salesmen.

A 2-approximation scheme for both the k-MST and the k-TSP given by Garg [24] is the best known approximation ratio.

If no penalty is considered, prize-collecting TSP is also referred to as quota-TSP problem [7, 8]. Awerbush [8] proposed an $O(\log^2 R)$ approximation algorithm where R is the quota (i.e., the amount of prizes to be collected). It is based on an approximation for the k-minimum-spanning-tree problem (k-MST), which is finding a tree of least weight that spans exactly k vertices on a graph. Ausiello et al. [7] studied the online version of the problem where requests are given over time.

However, all of above prize-collecting or quota TSP assumes there is only one traveling salesman. If multiple traveling salesmen are allowed, the prize-collecting multi-TSP has been studied under the name of multi-vehicle routing with profit, to which Chapter 9 in [21] gave a comprehensive survey. All the existing work, however, either fixed the number of vehicles or fixed the starting and ending depots of the vehicles. In contrast, in DAO^2, not only do we need to decide the number of initiators, but also to determine the starting and ending node of each initiator, thus differs dramatically from existing research. Our work is the first one to study multi-TSP prize collecting problem wherein both number of vehicles and their starting and ending nodes are not given.

In sensor network community, there are extensive research that focused on disconnection-tolerant operations in the absence of the base station. Some system research were conducted to design cooperative distributed storage systems and to improve the utilization of the network's data storage capacity [35, 36]. Other research instead took an algorithmic approach by focusing on the optimality of the solutions [27, 45, 52]. However, all above works assumed that there is enough storage space available to store the overflow data, thus not addressing the overall storage overflow problem.

Intermittently connected sensor networks are different from delay tolerant sensor networks (DTSN) [32]. In DTSNs, mobile nodes are intermittently connected with each other due to their mobility and low density, and data is opportunistically forwarded to destination nodes. In intermittently connected sensor networks, however, all the static sensors are connected with each other while being disconnected from the base station, and data is uploaded to the base station only when uploading opportunities such as data mules become available.

There is vast amount of literature of data aggregation in sensor networks [4, 15, 29, 31, 34, 47]. Tree-based routing structures were often proposed to either maximize network lifetime (the time until the first node depletes its energy) [34], or minimize total energy consumption or communication cost [29, 31], or reduce delay of data gathering [4]. Recently, Chen et al. [15] considered the duty-cycle sensor networks and designed two distributed data aggregation algorithms where aggregation tree and a conflict-free schedule are generated simultaneously to achieve low aggregation latency. Some other works were based on non-tree routing structures, using mobile base stations to collect aggregated data in order to maximize the network lifetime [42, 47]. Data aggregation in DAO^2, however, significantly differs from existing data aggregation techniques in both goals and techniques. First, existing data aggregation is to reduce number of transmissions by combining data from different sensors en route to base station, thus saving energy. The goal of data aggregation in DAO^2, however, is to aggregate the overflow data so that they can fit into storage available in the network, thus preventing data loss caused by overall storage overflow. Second, the underlying routing structures in most of the existing data aggregation techniques are trees rooted at the base station covering all sensor nodes. In DAO^2, however, since the base station is not available, those routing structures are no longer suitable. Instead, DAO^2 introduces minimum q-edge

forest, a routing structure that serves as the building block of our techniques.

VIII. Conclusion and Future Work

This paper introduced DAO^2, an architectural and algorithmic framework that tackles the overall storage overflow problem in intermittently connected sensor networks. We modeled the DAO^2 as a new graph-theoretic problem called multiple traveling salesmen selection and routing, and designed a suite of energy-efficient optimal, approximation, heuristic, and distributed data aggregation algorithms to solve it. The building block of our algorithmic techniques is minimum q-edge forest, a routing structure that generalizes minimum spanning tree and achieves energy-efficient data aggregation with performance guarantees. Because of this general and theoretical root, the techniques proposed in this paper could possibly be applicable to any applications wherein data correlation and resource constraints coexist.

As it is an architectural framework, there are a few future extensions. We will consider that overflow data generated from different data nodes could have different sizes, and that different storage nodes could have different storage capacities. We will also consider that different data nodes could have different correlation coefficients. Currently data aggregation and data offloading are treated as two separate stages – as ongoing [2] and future work, we will integrate these two stages and explore a more unified energy-efficient solution for the overall storage overflow problem.

Acknowledgment

This work was supported in part by NSF Grant CNS-1419952. We thank Yan Ma and Basil Alhakami for many discussions and simulations.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[2] B. Alhakami, B. Tang, J. Han, and M. Beheshti. Dao-r: Integrating data aggregation and offloading in sensor networks via data replication. In *Proc. of GLOBECOM*, 2015.

[3] B. Alhakami, B. Tang, J. Han, and M. Beheshti. Dao-r: Integrating data aggregation and offloading in sensor networks via data replication. *International Journal of Sensor Networks*, 29(2):134 – 146, 2019.

[4] B. Alinia, M. Hajiesmaili, and A. Khonsari. On the construction of maximum-quality aggregation trees in deadline-constrained wsns. In *Proc. of INFOCOM*, 2015.

[5] A. Archer, M. H. Bateni, M. T. Hajiaghayi, and H. Karloff. Improved approximation algorithms for prize-collecting steiner tree and tsp. In *IEEE FOCS*, 2009.

[6] C. Archetti, M. G. Speranza, and D. Vigo. Vehicle routing problems with profits. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, page 273–297, 2014.

[7] G. Ausiello, M. Demange, L. Laura, and V. Paschos. Algorithms for the on-line quota traveling salesman problem. *Information Processing Letters*, 92:89–94, 2004.

[8] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM J. Comput.*, 28(1):254–262, February 1999.

[9] B. Awerbuch, A. Bar-Noy, and M. Gopal. Approximate distributed bellman-ford algorithms. *IEEE Transactions on Communications*, 42:2515–2517, 1994.

[10] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[11] S. Basagni, L. Boloni, P. Gjanci, C. Petrioli, C. A. Phillips, and D. Turgut. Maximizing the value of sensed information in underwater wireless sensor networks via an autonomous underwater vehicle. In *Proc. of INFOCOM*, 2014.

[12] T. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Elsevier Omega*, 34:209–219, 2006.

[13] D. Bienstock, M. X. Goemans, and D. Simchi-Levi D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.

[14] C. Busch, M. Magdon-Ismail, F. Sivrikaya, and B. Yener. Contention-free mac protocols for wireless sensor networks. In *Proc. of DISC*, 2004.

[15] Q. Chen, H. Gao, Z. Cai, L. Cheng, and J. Li. Distributed low-latency data aggregation for duty-cycle wireless sensor networks. *IEEE/ACM Transactions on Networking*, 26(5):2347–2360, 2018.

[16] Y. Choi, M. Khan, V. A. Kumar, and G. Pandurangan. Energy-optimal distributed algorithms for minimum spanning trees. *IEEE Journal on Selected Areas in Communications*, 27(7):1297–1304, 2009.

[17] T. Corman, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009.

[19] R. Cristescu, B. Beferull-Lozano, M. Vetterli, and R. Wattenhofer. Network correlated data gathering with explicit communication: Np-completeness and algorithms. *IEEE/ACM Transactions on Networking*, 14:41–54, 2006.

[20] M. Dell'Amico, F. Maffioli, and A Sciomachen. A lagrangian heuristic for the prize collecting travelling salesman problem. *Annals of Operations Research*, 81:289–306, 1998.

[21] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188 – 205, 2005.

[22] M. Di Francesco, S. K. Das, and G. Anastasi. Data collection in wireless sensor networks with mobile elements: A survey. *ACM Trans. Sen. Netw.*, 8(1):7:1–7:31, 2011.

[23] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[24] N. Garg. Saving an epsilon: A 2-approximation for the k-mst problem in graphs. In *Proc. STOC '05*, 2005.

[25] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proc. of HICSS*, 2000.

[26] J.A. Hoogeveen. Analysis of christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10:291 – 295, 1991.

[27] X. Hou, Z. Sumpter, L. Burson, X. Xue, and B. Tang. Maximizing data preservation in intermittently connected sensor networks. In *Proc. of MASS*, 2012.

[28] S. Hsu, Y. Yu, and B. Tang. dre^2: Achieving data resilience in wireless sensor networks: A quadratic programming approach. In *Proc. of MASS 2020*.

[29] T. Kuo and M. Tsai. On the construction of data aggregation tree with minimum energy cost in wireless sensor networks: Np-completeness and approximation algorithms. In *Proc. of INFOCOM*, 2012.

[30] H. Li, D. Liang, L. Xie, G. Zhang, and K. Ramamritham. Flash-optimized temporal indexing for time-series data storage on sensor platforms. *ACM Trans. Sen. Netw.*, 10(4):1565–1572, 2012.

[31] J. Li, A. Deshpande, and S. Khuller. On computing compression trees for data collection in wireless sensor networks. In *Proc. of INFOCOM*, 2010.

[32] Y. Li and R. Bartos. A survey of protocols for intermittently connected delay-tolerant wireless sensor networks. *Journal of Network and Computer Applications*, 41:411–423, 2014.

[33] L. Liu, R. Wang, D. Guo, and X. Fan. Message dissemination for throughput optimization in storage-limited opportunistic underwater sensor networks. In *Proc. of SECON*, 2016.

[34] D. Luo, X. Zhu, X. Wu, and G. Chen. Maximizing lifetime for the shortest path aggregation tree in wireless sensor networks. In *Proc. of INFOCOM*, 2011.

[35] L. Luo, Q. Cao, C. Huang, L. Wang, T. Abdelzaher, and J. Stankovic. Design, implementation, and evaluation of enviromic: A storage-centric audio sensor network. *ACM Transactions on Sensor Networks*, 5(3):1–35, 2009.

[36] L. Luo, C. Huang, T. Abdelzaher, and J. Stankovic. Envirostore: A cooperative storage system for disconnected operation in sensor networks. In *Proc. of INFOCOM*, 2007.

[37] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[38] D. Mosse and G. Gadola. Controlling wind harvesting with wireless sensor networks. In *Proc. of IGCC*, 2012.

[39] L. Mottola. Programming storage-centric sensor networks with squirrel. In *Proc. of IPSN*, 2010.

[40] D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.

[41] F. Shahzad. Satellite monitoring of wireless sensor networks. *Procedia Computer Science*, 21:479 – 484, 2013.

[42] Y. Shi and Y.T. Hou. Theoretical results on base station movement problem for sensor network. In *Proc. of INFOCOM*, 2008.

[43] R. Sugihara and R. K. Gupta. Path planning of data mules in sensor networks. *ACM Trans. Sen. Netw.*, 8(1):1:1–1:27, 2011.

[44] B. Tang. dao^2: Overcoming overall storage overflow in intermittently connected sensor networks. In *Proc. of IEEE INFOCOM 2018*.

[45] B. Tang, N. Jaggi, H. Wu, and R. Kurkal. Energy efficient data redistribution in sensor networks. *ACM Transactions on Sensor Networks*, 9(2):11:1–11:28, May 2013.

[46] L. Tang and X. Wang. An iterated local search heuristic for the capacitated prize-collecting travelling salesman problem. *Journal of the Operational Research Society*, 59:590–599, 2008.

[47] S. Tang, J. Yuan, X. Li, Y. Liu, G. Chen, M. Gu, J. Zhao, and G. Dai. Dawn: Energy efficient data aggregation in wsn with mobile sinks. In *Proc. of IWQoS*, 2010.

[48] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2001.

[49] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315 – 332, 2016.

[50] L. A. Villas, A. Boukerche, H. de Oliveira, R. B. de Araujo, and A. A.F. Loureiro. A spatial correlation aware algorithm to perform efficient data collection in wireless sensor networks. *Ad Hoc Networks*, 12:69–85, 2014.

[51] B. Weiss, , H.L. Truong, W. Schott, A. Munari, C. Lombriser, U. Hunkeler, and P. Chevillat. A power-efficient wireless sensor network for continuously monitoring seismic vibrations. In *Proc. of SECON*, 2011.

[52] X. Xue, X. Hou, B. Tang, and R. Bagai. Data preservation in intermittently connected sensor networks with data priorities. In *Proc. of SECON*, 2013.

[53] H. Zheng and J. Wu. Data collection and event detection in the deep sea with delay minimization. In *Proc. of SECON*, 2015.

[54] J. Zheng and P. Wang C. Li. Distributed data aggregation using slepianwolf coding in cluster-based wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 59:2564 – 2574, 2010.

APPENDICS

Java Implementation
https://github.com/hungngo0309/SensorNetwork