DAO^2: Overcoming Overall Storage Overflow in Intermittently Connected Sensor Networks

Bin Tang, Hung Ngo, Yan Ma, and Basil Alhakami
Computer Science Department, California State University Dominguez Hills
btang@csudh.edu, hung.ngo.tm@gmail.com, hi.yanma@gmail.com, balhakami1@toromail.csudh.edu

***Abstract*—Many emerging sensor network applications operate in challenging environments wherein the base station is unavailable. Data generated from such *intermittently connected sensor networks* (ICSNs) must be stored inside the network for some unpredictable time before uploading opportunities become available. Consequently, sensory data could overflow the limited storage capacity available in the entire network, making discarding valuable data inevitable. To overcome such *overall storage overflow* in ICSNs, we propose and study a new algorithmic framework called *data aggregation for overall storage overflow* (DAO^2). Utilizing spatial data correlation that commonly exists among sensory data, DAO^2 employs data aggregation techniques to reduce the overflow data size while minimizing the total energy consumption in data aggregation. At the core of our framework are two new graph theoretical problems that have not been studied. We refer to them as *traveling salesmen placement problem* (TSP^2) and *quota traveling salesmen placement problem* (Q-TSP^2). Different from the well-known multiple traveling salesman problem (mTSP) and its variants, which mainly focus on the routing of multiple salesmen initially located at fixed locations, TSP^2 and Q-TSP^2 must decide the placement as well as the routing of the traveling salesmen. We prove that both problems are NP-hard and design approximation, heuristic, and distributed algorithms. Our algorithms outperform the state-of-the-art data aggregation work with base stations by up to 71.8% in energy consumption.**

Keywords – Sensor Networks, Data Aggregation, Approximation and Distributed Algorithms, Graph Theory

I. Introduction

Overall Storage Overflow. Sensor networks have been deployed to tackle some of the most fundamental problems facing human beings, such as disaster warnings, climate change, and renewable energy. These emerging scientific applications include underwater or ocean sensor networks [9, 22, 28, 35, 43, 60], wind and solar harvesting [34, 40], seismic sensor networks [42, 55], and monitoring of volcano eruption and glacial melting [18, 47]. 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.

It is usually impossible to deploy high-power, high-storage data-collecting base stations in those challenging environments. Consequently, large amounts of generated sensory data are stored inside the network for some unpredictable period of time and then collected by periodic visits of data mules or robots [27, 56, 59], or by low-rate satellite links [44, 57]. We refer to such sensor networks without the base station as *intermittently connected sensor networks* (ICSNs). In inhospitable environments, ICSNs must operate more resiliently than traditional sensor networks wherein base stations are always available.

In this paper, we focus on how to achieve data resilience in ICSNs. *Data resilience* refers to the long-term viability and availability of data despite insufficiencies of (or disruptions to) the physical infrastructure that stores the data. In ICSNs, one such disruption is sensor storage overflow. On one side, sensing a wide range of physical properties in the real world, above scientific applications generate massive amounts of data, such as videos or high-resolution images [27]. On the other side, storage is still a severe resource constraint of sensor nodes despite the advances in energy-efficient flash storage [35]. Consequently, the massive sensory data could overflow the data storage of sensor nodes and cause data loss. Such storage overflow problem is further exacerbated in ICSNs, wherein the high-storage base stations are not available to collect and store the data.

To avoid data loss, our previous works have designed a suite of techniques to *offload* overflow data from storage-depleted sensor nodes to nearby sensor nodes with available storage [25, 26, 48, 49, 58]. 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. Unfortunately, any of the existing data offloading techniques cannot alleviate this. We refer to this newly identified obstacle in the ICSNs as *overall storage overflow*. Below we give a more concrete example.

Motivating Example. Consider a recent application of underwater exploration and monitoring [9, 28], where camera sensors take pictures of the underwater scenes while an autonomous underwater vehicle (AUV) is dispatched periodically to collect the pictures from the sensors. Suppose there are 100 underwater camera sensors, 10 of which generate one 640×480 JPEG color image per second. Even using the latest 16 GB parallel NAND flash sensor storage [31], it takes less than one day to exhaust the storage of all the 100 camera sensors, causing an overall storage overflow. If the AUV cannot be dispatched timely due to inclement weather, discarding valuable data becomes inevitable. We thus answer the following question: *How to preserve the large amounts of data in ICSNs despite the overall storage overflow?*

Contributions. To overcome overall storage overflow, we propose to utilize spatial correlation that commonly exists among sensory data [54] and employ data aggregation techniques to reduce the overflow data size. The spatial correlation of sensory data is due to the proximity of sensor nodes detecting the same event of interest, thus producing data of similar values. For example, the partial overlapping between scenes from different cameras could produce similar and redundant images in the above underwater exploration scenario. We create a new algorithmic framework called *data aggregation for overall storage overflow* (DAO^2). At the core of DAO^2 are two fundamental graph-theoretical problems called *traveling salesmen placement problem* (TSP^2) and *quota traveling salesmen placement problem* (Q-TSP^2). Unlike the classic multiple traveling salesman problem (mTSP) and its variants [10, 52], before finding the routing for each traveling salesman, TSP^2 and Q-TSP^2 need to decide first how many of them are needed and where to place them. This makes the problems more general and more challenging than the classic mTSP. To our knowledge, both TSP^2 and Q-TSP^2 are not studied before.

To solve DAO^2, we design a suite of energy-efficient optimal, approximation, heuristic, and distributed data aggregation algorithms with rigorous performance guarantee analyses. One novelty of our aggregation techniques is two graph structures uniquely derived from the DAO^2 called *aggregation network* and *minimum q-edge forest*, where q is the number of sensor nodes that aggregate their overflow data (or the number of cities to visit in the TSP jargon). The minimum q-edge forest is a set of aggregation trees of total q cycle-less edges. It generalizes the minimum spanning tree, one of the most fundamental graph structures, and accurately captures the information needed for energy-efficient data aggregation.

After being aggregated to the size accommodable by the network, the overflow data can then be stored in sensor nodes with available storage using techniques proposed in [25, 26, 49, 58] (see Example 1 in Section II). Note that we do not consider how to upload data from sensor nodes to the base station, which has been studied extensively using data mules or robots [20, 27, 46, 56, 59]. In our conference paper [48], we solved DAO^2-U, a special and uniform case of DAO^2 where all the data nodes have the same overflow data size and the same correlation coefficient, and all the storage nodes have the same storage capacity. Our main contributions and paper organizations are as follows.

1). We identify and formulate a new algorithmic framework called DAO^2 that tackles the overall storage overflow problem in ICSNs. (Section I and II)

2). We show that DAO^2-U is equivalent to TSP^2, which is NP-hard. We design a suite of optimal, approximation, and heuristic algorithms. In particular, the approximation algorithm achieves $(2 - \frac{1}{q})$ approximation ratio, where q is the number of aggregators. (Section III)

3). We show that DAO^2 is equivalent to Q-TSP^2 and again design a suite of optimal, approximation, and heuristic algorithms. In particular, the approximation algorithm has an approximation ratio of $\log^2 \mathcal{Q}$, where $\mathcal{Q}$ is the targeted total data size reduction. (Section IV)

4). We design a suite of distributed data aggregation algorithms for the DAO^2 with performance guarantees and time and message analyses. (Section V)

5). Our algorithms outperform the existing data aggregation work with the base station by up to 71.8% in energy consumption. (Section VI)

II. Problem Formulation of DAO^2

This section introduces the DAO^2 and its network, data spatial correlation, and energy models. We then formulate the problem and illustrate it with an example.

Problem Statement of DAO^2. Fig. 1 illustrates the DAO^2. Some sensor nodes are close to the events of interest and thus are constantly generating sensory data and have depleted their storage. Such sensor nodes with depleted storage spaces while still generating data are **data nodes**. The newly generated data that can no longer be stored at a data node is **overflow data**. To avoid data loss, overflow data must be offloaded to sensor nodes with available storage (referred to as **storage nodes**). However, the total size of the overflow data is larger than the total available storage from the storage nodes, causing an overall storage overflow. Aggregating the overflow data is needed before offloading them to storage nodes.

In DAO^2, one or more data nodes will be selected as the **initiators**, which start the aggregation process by sending their overflow data to visit other data nodes in a multi-hop manner. When a non-initiator data node receives the data, it is made aware of the spatial correlation of the data and becomes an **aggregator** (refer to Spatial Correlation Data Model for more details). That is, it aggregates its own overflow data and then forwards the initiator's entire overflow data to another data node. This continues until enough data nodes become aggregators such that the total size of the overflow data in the ICSN after aggregation equals to or is slightly less than the total available storage in the ICSN. The goal of DAO^2 is to minimize the energy consumption in this process by finding the initiators and aggregators and the routes among them. Note that when a storage node receives any data, it relays it in its entirety as the entire data spatial correlation information is needed for data aggregation.

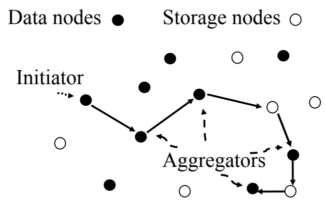


Fig. 1. Illustrating DAO^2.

Network Model. The ICSN 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, where data node $i \in V_d$ has R_i bits of overflow data. The rest $|V| - p$ sensor nodes are storage nodes, where storage node $j \in V - V_d$ has m_j bits of available storage space. Due to the overall storage overflow, $\sum_{i \in V_d} R_i > \sum_{j \in V - V_d} m_j$. Let $\mathcal{Q}$ denote the 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 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 as $H(i|j_1, ..., j_p) = R_i$ 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 other data node, its overflow data is entropy coded as $H(i|j_1, ..., j_p) = r_i \leq R_i$.

We denote $\rho_i = 1 - r_i/R_i$ as the *correlation coefficient* of data node i, thus $R_i \cdot \rho_i$ is the amount of data size reduction at i after aggregation when it receives an initiator's data. ρ_i indicates the data redundancy level at data node i. $\rho_i = 0$ means i's data is entirely different from others and thus cannot be aggregated; $\rho_i = 1$ means i's data is a duplicate copy of others' data and therefore can be obliterated. Note that given any instance of overall storage overflow, it must be $\sum_{i \in V_d}(R_i \cdot \rho_i) \geq \mathcal{Q}$ for a *feasible data aggregation*; i.e., the data aggregation can achieve the targeted data reduction $\mathcal{Q}$ with the given ρ_i.

The well-known joint entropy-based coding model inspires our model in [16].[1] This model, however, assumes that different sensor nodes not only have the same amount of data but also have the same level of data correlations. In a real scenario, usually, the farther away a sensor node is from the event of interest, the less data it generates; the closer of two sensor nodes, the higher similarity of their generated data. With varying data sizes and varying correlation coefficients, our model is thus more realistic and more general. We make two assumptions.

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

Assumption 2: Each aggregator i can be visited multiple times by the same or different initiators (if that is more energy-efficient). However, its data can only be aggregated once, reducing the size from R_i to r_i. This is because i's data reduction is based on its spatial correlation coefficient ρ_i, not how often initiators visit it. □

Energy Model. We adopt the first-order radio model [23] for the battery power consumption of sensor nodes. 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 the energy consumption per bit on transmitter and receiver circuits, and $\epsilon_{amp} = 100pJ/bit/m^2$ is the 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 $w(R_u, u, v) = E_t(R_u, l_{u,v}) + E_r(R_u) = R_u \times (2 \cdot E_{elec} + \epsilon_{amp} \times l_{u,v}^2)$, and $c(R_u, W) = \sum_{i=1}^{n-1} w(R_u, v_i, v_{i+1})$ be the *aggregation cost* on W, the energy consumption of sending R_u-bit from v_1 to v_n along W. We assume a contention-free MAC protocol exists to avoid overhearing and collision (e.g., [12]).

[1]We are aware of other distributed coding techniques such as Slepian-Wolf coding [61]. In their model, each node has R bits of data, which can be reduced to $r \leq R$ bits when the node receives side information from others. However, they need a global correlation structure and thus are impractical for large networks.

Problem Formulation of DAO^2. The goal of DAO^2 is to find a set of a $(1 \leq a < p)$ initiators $\mathcal{I}$ 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 $I_j \in \mathcal{I}$, s.t. the total amount of data reduction $\sum_{i \in A}(R_i \cdot \rho_i) >= \mathcal{Q}$ while minimizing the *total aggregation cost* $\sum_{1 \leq j \leq a} c(R_{I_j}, W_j)$. Here, $A = \bigcup_{j=1}^{a}\{W_j - \{I_j\} - G_j\}$ is all the aggregators being visited and G_j is the set of storage nodes in W_j. Table I lists all the notations used in different problems.

TABLE I
SUMMARY OF THE NOTATIONS.

	DAO^2
$G(V, E)$	The ICSN graph
V_d and p	The set and number of data nodes, where $V_d \subset V$
R_i	Overflow data size at data node i before aggregation
$r_i \leq R_i$	Overflow data size at data node i after aggregation
ρ_i	Correlation coefficient at data node i, $\rho_i = 1 - r_i/R_i$
m_j	Storage capacity of a storage node $j \in V - V_d$
$\mathcal{Q}$	$\mathcal{Q} = \sum_{i \in V_d} R_i - \sum_{j \in V - V_d} m_j$, total data reduction
$\mathcal{I}$, a	Set and number of initiators, $1 \leq a < p$
I_j	j^{th} initiator, $1 \leq j \leq a$
W_j	Aggregation walk or path starting with I_j
$w(R_u, u, v)$	Aggregation cost of sending R_u bits from u to v
$c(R_u, W_j)$	Aggregation cost of sending R_u bits along W_j
	DAO^2-U
R	Overflow data size at a data node before aggregation
r, $r \leq R$	Overflow data size at a data node after aggregation
m	Storage capacity of a storage node in $V - V_d$
q	Number of aggregators needed
	TSP^2 and Q-TSP^2
$G(V', E')$	Aggregation network for TSP^2 and Q-TSP^2
$w(u, v)$	Weight of an edge $(u, v) \in E'$
$d(u, v)$	Length of the shortest path between nodes $u, v \in V'$
pr_i	Prize available at node $i \in V'$
$mode_i$	Traveling mode (cost per traveled distance) at node i
$\mathcal{Q}$	Target prize to collect
$pcr(C_i, C_j)$	Prize-cost ratio of two fragments 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 TSP^2 and Q-TSP^2

EXAMPLE 1: Fig. 2 gives an example of DAO^2 in a grid ICSN of 9 nodes (we use the grid only for illustration purposes). Nodes B, D, E, G, and I are data nodes, with $R_B = R_E = 2$ and $R_D = R_G = R_I = 1$. Nodes A, C, F and H are storage nodes, with $m_j = 1$ for all of them except that $m_A = 2$. The energy cost on any edge is 1 for one data unit, $\rho_i = 1/2$ at any data node i. Overall storage overflow arises as there are seven units of overflow data but only five units of storage spaces, giving $\mathcal{Q} = 2$. The optimal solution, shown in the blue arrowed line, selects D as the initiator and sets its aggregation path as D, E, and B, with a 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 two 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 be stored in the five storage spaces, solving the overall overflow problem. □

Data Offloading After Data Aggregation. After aggregation, data is offloaded from data nodes to storage nodes with minimum energy consumption. Our previous work [26, 49] has shown this can be modeled as a minimum cost flow problem [1], which can be solved optimally and efficiently. One optimal solution in Fig. 2 is offloading the two units of data at B to A, E's 1 unit data to C, G's 1 unit data to H, and I's 1 unit of data to F, resulting in an offloading cost of six. In this paper, we only focus on data aggregation as data offloading can be achieved optimally, and leave integrating them into a more unified energy-efficient solution as future work.

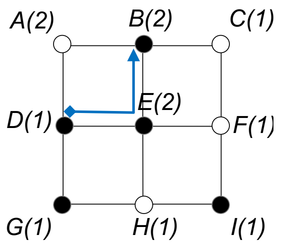


Fig. 2. An example for DAO^2. Numbers in the parentheses are R_i (for data nodes ●) and m_j (for storage nodes ○).

III. Algorithmic Solutions for DAO^2-U

In this section, we study a special and uniform case of DAO^2, referred to as DAO^2-U. In DAO^2-U, all the data nodes have the same overflow data size and correlation coefficient (i.e., $R_i = R$, $r_i = r$, and $\rho_i = \rho = 1 - r/R$, $i \in V_d$) and all the storage nodes have the same storage capacity (i.e., $m_j = m$, $j \in V - V_d$).

A. Problem Formulation of DAO^2-U

We first derive the valid range of p for the occurrence of both overall storage overflow and feasible aggregation. The overall storage overflow condition gives $p \times R > (|V| - p) \times m$, thus $p > \frac{|V|m}{m+R}$. Denote the number of aggregators needed as q. 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

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

Next, we compute the upper bound of p for feasible data aggregation. As at least one data node needs to be the initiator to start the aggregation process, there can only be a maximum of $p-1$ aggregators (Assumption 1). We therefore have $q = \lceil \frac{p\times(R+m)-|V|\times m}{R-r} \rceil \le p - 1$, which gives $p \le \lfloor \frac{|V|m-R+r}{m+r} \rfloor$. The valid range of p for the occurrence of both overall storage overflow and feasible aggregation is therefore

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

Given a valid p value and its corresponding q value, meaning q data nodes are aggregators and the rest $p-q$ data nodes *can* be initiators (Assumption 1), DAO^2-U is to determine a set of a $(1 \le a \le (p-q))$ initiators $\mathcal{I}$ and a corresponding set of a *aggregation walks*: W_1, W_2, ..., W_a, where W_j $(1 \le j \le a)$ starts from a distinct initiator $I_j \in \mathcal{I}$, such that $|\bigcup_{j=1}^{a}\{W_j - \{I_j\} - G_j\}| = q$ aggregators are visited while the *total aggregation cost* in this process $\sum_{1 \le j \le a} c(R, W_j)$ is minimized.

EXAMPLE 2: Fig. 3 shows the same ICSN in Fig. 2. For DAO^2-U, lets assume $R = m = 1$ and $\rho = 3/4$. Overall storage overflow exists as there are four units of storage space while five units of overflow data. The number of aggregators $q = 4$ following Equation 1, leaving one data node as the initiator. One optimal solution is selecting B as the initiator and setting its aggregation path as B, E, D, G, H, and I, as shown in the blue arrowed line. It has a 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 a total of 4 units and thus can be offloaded to storage nodes using techniques in [26, 49]. Note that 7/4 units of data at I now include 3/4 units of I's own aggregated overflow data and one unit of initiator B's overflow data. □

DAO^2-U gives rise to a new graph-theoretical problem, referred to as *traveling salesmen placement problem (TSP^2)*. Next, we formulate TSP^2, prove its NP-hardness, and design a $(2 - \frac{1}{q})$-approximation algorithm. We then prove that the DAO^2-U in an ICSN is equivalent to the TSP^2 in a properly transformed graph of the ICSN, called *aggregation network*. Therefore the algorithms for TSP^2 can be applied to solve DAO^2-U.

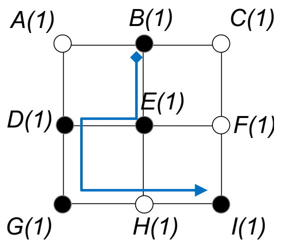


Fig. 3. An example for DAO^2-U. Numbers are R_i (for data nodes ●) and m_j (for storage nodes ○). The blue arrowed line shows the aggregation path.

B. Traveling Salesmen Placement Problem (TSP^2)

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 TSP^2 is to determine a set of *at most* $|V'| - q$ *starting nodes*, at each of which a salesman is placed and then starts 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 the weight of edge $(u, v) \in E'$. We assume that triangle inequality holds: for edges $(x, y), (y, z), (z, x) \in E'$, $w(x, y) + w(y, z) \ge w(z, x)$. 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 TSP^2 is to decide:

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

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

Theorem 1: The TSP^2 is NP-hard.

Proof: Given an undirected graph $G'(V', E')$, its *metric completion* $G^{mc}(V', E^{mc})$ 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 $G'(V', E')$. When $q = |V'| - 1$, TSP^2 in G^{mc} becomes how to find a minimum-cost *hamiltonian path* that visits each node in G^{mc} exactly once. This is the "without fixed endpoints" version of *traveling salesman path problem*

(TSPP) [24], which finds a minimum-cost *hamiltonian path* that visits each node in G^{mc} exactly once. Thus TSPP is a special case of TSP^2. Below we prove TSPP is NP-hard by reducing the well-known *traveling salesman problem (TSP)* [15] to TSPP. Recall that TSP is to find a minimum-cost *Hamiltonian cycle* in a complete graph that visits each node exactly once.

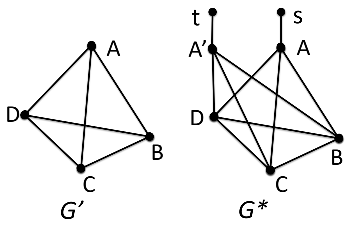


Fig. 4. Proving TSPP is NP-hard.

As shown in Fig. 4, let complete graph G' be an instance of TSP, and 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 in G' except A and assign the same cost on each edge as the corresponding edge in G' (i.e., $(A^{'}, B)$ has the same cost as (A, B), 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. 4) and assign their costs to be *infinite*. We show that G' contains a minimum-cost Hamiltonian cycle if and only if G^* contains a minimum-cost Hamiltonian path.

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 $A^{'}$. Instead of returning to $A^{'}$, the resultant path returns to A, forming a minimum-cost Hamiltonian cycle in G'. ■

2) Approximation Algorithm for TSP^2: We introduce some definitions before presenting the approximation algorithm.

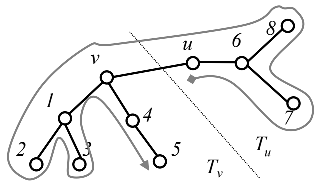
Definition 1: (*Binary Walk (B-Walk), q-Edge Forest*) Given a tree $T \subset G'$ with a maximum-weight edge (u, v) (ties are broken randomly), T is divided into (u, v) and subtrees T_u and T_v. The B-walk on T, denoted as $W_B(T)$, starts from u and visits all the nodes in T_u following depth-first-search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

A forest F of G' is a subgraph of G' that is acyclic (and possibly disconnected). A q-edge forest, denoted as F_q, is a forest with q edges. □

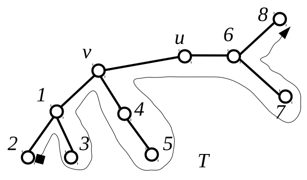
Fig. 5(a) shows a tree T with $w(u, v) = 2$ and weights of other edges being 1, and a B-walk of cost 16. In B-walk, each edge in T_u is traversed twice, and each edge in T_v is traversed once or twice. B-walk saves cost traversing a tree since the maximum-weight (u, v) is traversed only once.

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 $c(W_B(T)) \leq \big(2 \times c(T) - \frac{1}{|T|} \times c(T)\big) = (2 - \frac{1}{|T|}) \times c(T)$. ■



(a) B-Walk.



(b) LP-Walk.

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

Approximation Algorithm. Algo. 1 works as follows. Lines 1 and 2 sort all the edges in E' in the non-descending order of their weights and initialize an empty edge set E_q. 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. If it is linear, a salesman is placed at one end of it and then visits the rest nodes exactly once; if it is a tree, u or v, a salesman is placed at u or v, where (u, v) is the maximum-weight edge, and then does a B-walk to visit all the nodes (lines 11-15).

Algorithm 1: Approximation Algorithm for TSP^2.

```
Input: $G'(V', E')$ and number of nodes to visit $q$;
Output: $a$ walks: $W_1, W_2, ..., W_a$, and $\sum_{1 \leq j \in a} c(W_j)$;
Notations: $E_q$: set of $q$ cycleless edges in $G'$;
   $G'[E_q]$: subgraph of $G'$ induced by $E_q$, 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.         $i$++;
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) Place a salesman at one end node
           of $C_j$ and visits the rest nodes in $C_j$ once;
13.     if ($C_j$ is a tree) Place a salesman at $u$ or $v$, where
           $(u, v)$ is the maximum-weight edge, and 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 [14], 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'. Therefore, Algo. 1 generalizes Kruskal's algorithm, and MST is a special case of q-edge forest. We show that $G'[E_q]$ is a *minimum q-edge forest* defined below.

Definition 2: (*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 contradiction, assume that another q-edge forest, O_q, is a minimum q-edge forest with a cost smaller than $G'[E_q]$. 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 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, $e_l^g = e_n, 1 \leq n \leq |E'|$. We have two subcases:

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 TSP^2 with a 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 $q' = \sum_{j=1}^{\lambda}(l_j - 1)$. Therefore, the subgraph induced by all T_j^o $(1 \leq j \leq \lambda)$ is 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-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.

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 once or twice. A simple improvement is to traverse, between T_u and T_v, the one with a smaller cost first. We refer to this special case of B-Walk as *smaller-tree-first-walk (STF-walk)*. Indeed Fig. 5(a) shows an STF-walk.

A Heuristic Algorithm. Next, we present a 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* defined below.

Definition 3: (*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, so every edge in P is traversed *exactly once*. □

In LP-walk, since edges in the longest path are traversed only once, the cost of a walk can be further reduced. Finding the longest path in a tree is to find the shortest path among all pairs of leaf nodes and choose the longest one, which takes $O(|V|^3)$. Fig 5(b) shows an LP-walk costing 14. Because the maximum-weight edge (u, v) is not necessarily on the longest path P, we cannot obtain a 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.

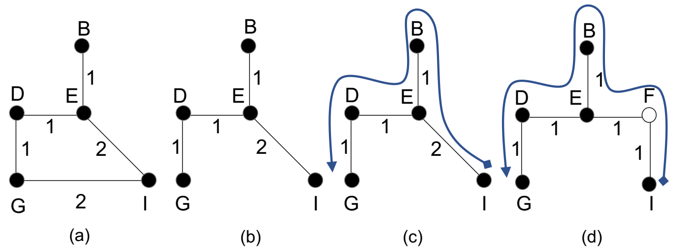


Fig. 6. (a) Aggregation network G' of ICSN G in Fig. 3. (b) 4-edge forest F_q resulted from Algo. 1. (c) B-walk on F_q. (d) Aggregation walk in G with aggregation cost of 6. The numbers on the edges are their weights.

C. Equivalency Between TSP^2 and DAO^2-U

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

Definition 4: (*Aggregation Network* $G'(V', E')$) V' is the set of p data nodes in V, $V' = V_d$. For any two data nodes $u, v \in V'$, an edge $(u, v) \in E'$ exists 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. □

After applying Algo. 1 on $G'(V', E')$, the resulting starting nodes in V' become the initiators in ICSN graph $G(V, E)$, and the rest nodes in V' become aggregators nodes in G.

EXAMPLE 3: Fig. 6(a) shows the aggregation network G' of the ICSN graph G in Fig. 2. 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, wherein I is selected as the initiator by Algo. 1, and the rest of the nodes (i.e., B, D, E, G) become aggregators. 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 an LP-walk. □

Note that the aggregation network G' is a new graph structure different from the *distance graph* in computing Steiner tree [29]. A distance graph of data nodes in G is a complete graph G_d with the weight of every edge $(u, v) \in G_d$ being the cost of the shortest path from u to v in G whereas an aggregation network does not need to be complete. Next, we prove that solving DAO^2-U in G is equivalent to solving TSP^2 in G'.

Theorem 3: DAO^2-U in the ICSN graph G is equivalent to TSP^2 in aggregation network graph G'.

Proof: It suffices to show that the data, energy consumption, and topology information used for computing energy-efficient aggregation in G are all mapped in G'. Below we show that this is achieved when transforming G to G'.

First, data information is preserved as all the data nodes in G are now nodes in G'. 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 an edge (X, Y), whose weight is the cost of any of such shortest paths. Therefore the energy consumption information is preserved. Third, if at least one shortest path exists 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 are on the same aggregation walk, they will visit other intermediate data nodes before visiting each other. Therefore, the topological requirement of DAO^2-U to "visit most data nodes (aggregators) with least energy cost" is achieved. Therefore, solving TSP^2 in G' is equivalent to solving DAO^2-U in G. ∎

IV. Algorithmic Solutions for DAO^2

In this section, we study the general DAO^2 with heterogenous overflow data sizes R_i, the data sizes after aggregation r_i, correlation coefficients ρ_i, and storage capacities m_j. We show that DAO^2 equals another new graph theoretical problem. We refer to it as the *quota traveling salesmen placement problem* (Q-TSP^2), which generalizes TSP^2 in the previous section. Below we formulate Q-TSP^2, prove its equivalency to DAO^2, and propose approximation and heuristic algorithms.

A. Quota Traveling Salesmen Placement Problem (Q-TSP^2)

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.[2] Besides, node i has a value of $mode_i$, which is a salesman's *traveling cost per unit distance* if he is placed and starts traveling at i.[3] Given a walk $W = \{v_1, v_2, ..., v_n\}$, the *traveling cost* on W by a salesman from node i is $c(i, W) = mode_i \cdot \sum_{j=1}^{n-1} w(v_j, v_{j+1})$. The objective of the Q-TSP^2 is to determine a) a set of *starting nodes* $\mathcal{I} \subset V'$, at $I_j \in \mathcal{I}$ a salesman is placed, 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 while the *total collected prizes* $\sum_{k \in A} pr_k \ge \mathcal{Q}$. Here, $A = \bigcup_{j=1}^{|\mathcal{I}|} \{W_j - \{I_j\}\}$ are all the nodes the salesmen visit and exclude their starting nodes. This is to be consistent with Assumption 1 in DAO^2 wherein the initiator's data cannot be aggregated; thus, no prize is to be collected at initiators (however, our solutions can be easily adjusted for the case that prizes can be collected at starting nodes).

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

Theorem 4: DAO^2 in the ICSN graph G is equivalent to Q-TSP^2 in aggregation network graph G'.

Proof: As Theorem 3 has shown that any topology-related information including data and energy costs is preserved in the transformation, we only need to show the prize-related information is mapped from G to G'. First, in a DAO^2 instance, an aggregator i's data reduction is $R_i \cdot \rho_i$, the prize pr_i a salesman collects when he visits i in a Q-TSP^2 instance. Second, in a DAO^2 instance, the total size of data reduction in G is $\mathcal{Q} = \sum_{i \in V_d} R_i - \sum_{j \in V - V_d} m_j$. In a Q-TSP^2 instance, this is indeed the targeted quota $\mathcal{Q}$ for the traveling salesmen to achieve in G'. Third, the aggregation costs in a DAO^2 instance depend on the sizes of the initiators' data packets, which corresponds in a Q-TSP^2 instance that salesmen dispatched from different starting nodes have different modes in terms of travel costs per unit distance. As any Q-TSP^2 instance corresponds to a DAO^2 instance, solving Q-TSP^2 in G' is equivalent to solving DAO^2 in G. ∎

B. Approximation Algorithm for Q-TSP^2

Before presenting Algo. 2, we first give a few definitions. We use $d(u, v)$ to denote the shortest path length between any two nodes $u, v \in V'$.

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

[3] For example, different nodes could have different transportation means available (cars, trains, and air), which incur different costs per unit distance.

Definition 5: (*Prize of a Fragment and Prize-Cost Ratio of Two Fragments*) The prize of a fragment (i.e., 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 fragments 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 C_i and 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 idea of Algo. 2 is to iteratively merge two fragments with the largest pcr until the total prize of all the fragments with at least two nodes (i.e., *combined fragments*) reaches $\mathcal{Q}$. It starts with n fragments, each is one node, and its ID is the node's ID. In each iteration, it combines two fragments with the largest pcr using the shortest path between them (ties are broken randomly) and updates the total prize (i.e., quota) collected so far (lines 3-9). It then takes the smaller ID as the ID of the new combined fragment and updates its prize as the sum of the prizes of both fragments and the prizes at the nodes on the connecting shortest path (lines 10-12). It also finds the starting node (with the smallest *mode* value) in the combined fragment, so its prize is excluded from the collected quota (lines 13-14). This continues until the combined fragments' total prize reaches $\mathcal{Q}$. Finally, a traveling salesman is placed at the starting node in each combined fragment, then visits all other nodes in the combined fragment to collect prizes by traversing each edge at most twice, and returns the total collected prizes and total traveling cost (lines 16-21). The time complexity of Algo. 2 is $O(\mathcal{Q} \cdot |V'|)$.

Algorithm 2: Approximation Algorithm for Q-TSP^2.
Input: $G(V', E')$, pr_u at node u, targeted quota $\mathcal{Q}$;
Output: total collected prizes $quota$, total traveling cost $\mathcal{T}$;
Notations: $quota$: prizes collected so far, initially zero;
$\mathcal{C}$: the set of all the fragments,
$\mathcal{C} = \{C_1, C_2, ..., C_{|V'|}\}$, initially $C_i = \{i\}$, $\forall i \in V'$;
A: IDs of the resultant combined fragments, initially empty;

```
1.  $quota = \mathcal{T} = 0$, $A = \phi$ (empty set);
2.  while ($quota \le \mathcal{Q}$)
        // $(C_{i^*}, C_{j^*})$ has the maximum $pcr$
3.      $(i^*, j^*) = \text{argmax}_{(i,j),\text{where } i,j \in A, i \neq j} pcr(C_i, C_j)$;
4.      $d(u,v) = d(C_{i^*}, C_{j^*})$, where $u \in C_{i^*}, v \in C_{j^*}$;
5.      $E(u,v)$ are all edges on the shortest path btw $u$, $v$;
6.      $N(u,v)$ are all the nodes on the shortest path
            between (and excluding) $u$, $v$, could be empty;
7.      if ($|C_{i^*}| == 1$) $quota$ += $pr(C_{i^*})$;
8.      if ($|C_{j^*}| == 1$) $quota$ += $pr(C_{j^*})$;
9.      $quota$ += $\sum_{i \in N(u,v)} pr_i$;
        // Merge into the fragment with smaller ID;
10.     $a = \min\{i^*, j^*\}$, $b = \max\{i^*, j^*\}$;
11.     $C_a = C_a \cup C_b \cup E(u,v)$, $A = A \cup \{a\}$;
12.     $pr(C_a)$ += $pr(C_b) + \sum_{i \in N(u,v)} pr_i$;
        // Starting node's prize is not in the collected quota
13.     $y = \text{argmin}_{u \in C_a} mode_u$;
14.     $quota$ −= $pr_y$;
15. end while;
        // $A$ includes the IDs of the combined fragments
16. for (each element $i \in A$)
17.     $s = \text{argmin}_{u \in C_i} mode_u$; // starting node in $C_i$
18.     Place a salesman at $s$, let $W_i$ denote the walk
            along which he visits all nodes in $C_i$ by
            traversing each edge at most twice;
19.     $\mathcal{T}$ += $c(s, W_i)$;
20. end for;
21. RETURN $quota$ and $\mathcal{T}$.
```

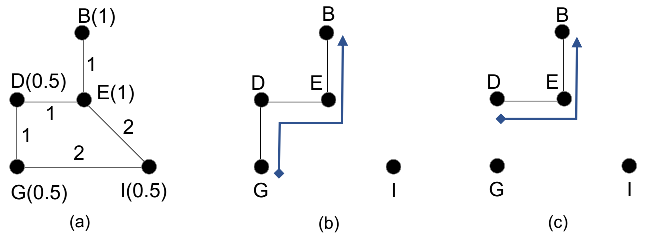


Fig. 7. (a) Aggregation network G' of ICSN G in Fig. 2. The numbers in the parentheses are the prizes available at data nodes. (b) and (c) are two possible solutions, with (c) being optimal. Algo. 2 outputs both (b) and (c) whereas Algo. 3 outputs only (c).

EXAMPLE 4: For the DAO^2 example in Fig. 2, the amount of data reduction at data node i is the prize pr_i available at node i in Q-TSP^2, and the total amount of data reduction to achieve is the total prize $\mathcal{Q}$ to collect. That is, $\mathcal{Q} = 2$, $pr_B = pr_E = 1$, and $pr_D = pr_G = pr_I = 0.5$. Next, we show how Algo. 2 solves this example. Fig. 7(a) shows its aggregation network. 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 a total prize of 2.5 with a 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 a total traveling cost of 2. □

When all the nodes have the same prizes and modes, Algo. 2 degenerates to Algo. 1. Algo. 2 is inspired by Awerbuch et al. [6] that solves a special case of Q-TSP^2. In particular, they considered a *quota-driven salesman problem* in which a single traveling salesman collects prizes in different cities to reach a target quota while minimizing the traveling cost. 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), finding a tree of the least weight that spans exactly k vertices on a graph. We thus give the below theorem without proof.

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

C. Heuristic Algorithm for Q-TSP^2

Algo. 2 iteratively combines two fragments that yield the maximum prize-cost ratio until prize quota $\mathcal{Q}$ is reached. One drawback of this approach is that the combined two fragments could have prizes much larger than the target $\mathcal{Q}$, thus costing more energy than necessary. For example, the solution in Fig. 7(b) collects prizes of 2.5, 25% more than target quota $\mathcal{Q} = 2$, and costs 3, 50% more than the optimal cost of 2 obtained in Fig. 7(c). We thus design a prize-collecting scheme viz. Algo. 3 that takes place on a local and more fine-grained level than Algo. 2, and shows that it constantly outperforms Algo. 2. We first give the below definition.

Definition 6: (*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 on 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 into the fragments until $\mathcal{Q}$ or a slightly higher amount of prizes is collected. There are three possible cases when adding a cycleless edge e_i: a) it starts a new combined fragment (lines 6-9), b) it connects two existing combined fragments (lines 10-13), or c) e_i merges with one existing combined fragment (lines 14-20). In each case, the starting node and the collected prizes in the newly combined fragment are updated accordingly. Next, the traveling salesman is placed at the node in each combined fragment with the smallest *mode* value. Finally, the salesman is dispatched to visit all other nodes to collect prizes by traversing each edge at most twice, with the total collected prizes and total cost returned (lines 24-28). The time complexity of Algo. 3 is $O(|E'|\log|E'| + \mathcal{Q})$.

Algorithm 3: Benefit-based Algorithm for Q-TSP^2.

```
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 prize-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 fragments;
     $\mathcal{C}$: the set of all the fragments,
         $\mathcal{C} = \{C_1, C_2, ..., C_{|V'|}\}$, initially $C_i = \{i\}, \forall i \in V'$;
     $A$: IDs of the resultant combined components;
     $ts_j$: the traveling salesman in component $C_j$, $1 \le j \le a$;
1.   $\mathcal{B}(e_1) \ge \mathcal{B}(e_2) \ge ... \ge \mathcal{B}(e_{|E|})$; // Sort edges in $\mathcal{B}$
2.   $i = 1$, $quota = 0$, $a = 0$, $A = E_{pc} = \phi$ (empty set);
3.   while ($quota \le \mathcal{Q}$) // process $e_i$ in decreasing order of $\mathcal{B}$
4.     Let $e_i = (n_1, n_2)$;
5.     if ($e_i$ causes a cycle w.r.t. $E_{pc}$) continue;
       // $e_i$ initiates a new fragment
6.     if ($sel(n_1) == sel(n_2) == false$)
7.       $sel(n_1) = sel(n_2) = true$;
         $quota$ += $(pr_{n_1} + pr_{n_2})$;
8.       if ($n_1 \le n_2$) $A = A \cup \{n_1\}$, $C_{n_1} = \{e_i\}$;
9.       else $A = A \cup \{n_2\}$, $C_{n_2} = \{e_i\}$;
       // $e_i$ merges two existing combined fragments
10.    elseif ($sel(n_1) == sel(n_2) == true$)
11.      Let $C_b$ and $C_c$ be $n_1$ and $n_2$'s belonged fragments;
         // Merge into the fragment with smaller ID
12.      if ($b \le c$) $A = A - \{c\}$, $C_b = C_b \cup C_c$;
13.      else $A = A - \{b\}$, $C_c = C_b \cup C_c$;
14.    else // $e_i$ merges with a combined fragment
15.      if ($sel(n_1) == false$) $x = n_1$;
16.      else $x = n_2$;    // $sel(n_2) == false$
17.      $quota$ += $pr_x$, $sel(x) = true$;
18.      Let the fragment $e_i$ merges with be $C_b$;
19.      if ($x \le b$) $A = A - \{b\} \cup \{x\}$;
20.    end else
21.    $E_{pc} = E_{pc} \cup \{e_i\}$;
22.    $i$++;
23.  end while;
24.  for (each element $i$ in $A$)
25.    $s = \text{argmin}_{u \in C_i} mode_u$; // starting node in $C_i$
       Place a salesman at $s$, who visits each node
         in $C_i$ by traversing each edge at most twice;
       Let the resultant walk be $W_i$;
26     $\mathcal{T}$ += $c(s, W_i)$;
27.  end for;
28.  RETURN  $quota$ and $\mathcal{T}$.
```

EXAMPLE 5: In Fig. 7(a), as $\mathcal{B}(B, E) \ge \mathcal{B}(D, E) \ge \mathcal{B}(D, G) \ge \mathcal{B}(E, I) \ge \mathcal{B}(G, I)$, 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). □

Discussions. Algo. 2 and 3 are mainly designed for one round of aggregation to achieve the performance approximation ratio. As such, it is based upon the assumption that after one round of aggregation and data offloading, the uploading opportunities will arrive timely to collect the data and empty the storage spaces of all the sensor nodes. Another round of sensing and aggregation will then take place. Otherwise, when both newly generated data and old aggregated data are present in the BSN, the data correlation model introduced in this paper can no longer work. The algorithms still work in a dynamic situation wherein different sensor nodes emerge as data nodes emerge. As long as the aggregated data can be collected timely by the uploading opportunities, the aggregation frequency can be kept up to speed with the arrival frequency of the uploading opportunities. The data aggregation process, triggered when the overflow data size is larger than the available storage space in the BSN, can always occur.

Algo. 1 for TSP^2 is a special case of Algo. 2 and 3 for Q-TSP^2. When all nodes have the same prizes, the largest prize-cost ratio of two fragments in Algo. 2 and the edge with the largest benefit in Algo. 3 degenerate to the cycless edge with the smallest weight in Algo. 1, and the resultant q-edge forest in Algo. 1 is indeed the merged fragments found in Algo. 2 and Algo. 3.

V. Distributed Algorithms

A. Distributed Algorithms for DAO^2-U

The distributed algorithm, referred to as *Distributed DAO^2-U*, consists of three stages. First, it constructs the aggregation network of the data nodes $G'(V', E')$ from the ICSN $G(V, E)$ by a modified distributed Bellman-Ford algorithm [39]. Second, the data nodes in the aggregation network cooperatively find the q-edge forest (i.e., a set of aggregation trees) based on a classic distributed MST algorithm [21, 41]. Third, an initiator is selected for each aggregation tree in the q-edge forest and starts the data aggregation process to reduce the overflow data size. Below we illustrate each stage in detail.

Stage 1: Constructing Aggregation Network. The distributed Bellman-Ford (DBF) algorithm [39] is a well-known asynchronous technique to compute the shortest paths between nodes in a network. Each node (data node or storage node) initially only has direct knowledge of its local links and sends messages about its perceived shortest path lengths (i.e., the routing table) to all other nodes to its neighbors. When a neighbor receives the message, if it finds that its cost to a node is greater than the sum of its cost to the sender and the sender's cost to that node, it updates its routing table and sends it to its neighbors. This takes place iteratively until all the nodes have the accurate shortest path information to other nodes in the network. The message size in DBF is $O(|V|)$, where $|V|$ is the number of nodes in the ICSN.

However, there are two challenges when we apply DBF to construct aggregation networks. First, it is well known that the message complexity of asynchronous DBF could be exponential [7]. This can be overcome by the fact that wireless communication is generally broadcast. Using broadcast, the message complexity of DBF is reduced to $O(|V|^2)$. Second, to construct an aggregation network on top of DBF, each data node needs to find if it has a "direct" link with another data node in the aggregation network. To do this, the message sent by any node includes the shortest paths to all other nodes and has a size of $O(|V|^2)$. After receiving this information, a data node checks its shortest path to each other data node; if there are no other data nodes on this shortest path, it has an edge over this data node in the aggregation network, and the cost of the edge is the cost of the corresponding shortest path.

Stage 2: Constructing q-edge Forest. Next, the p data nodes cooperate to find a q-edge forest in the aggregation network in a distributed manner. We propose two algorithms: a baseline algorithm and a fragment-based algorithm.

Baseline Algorithm. The baseline algorithm works as follows. At the end of aggregation network construction, each data node knows the IDs of other data nodes and the shortest paths to them. The node with the smallest ID is thus selected as the leader. Then every node sends the weights of all its incident edges to the leader, following the shortest path between them. 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 the 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 the computed edges; if so, it marks these edges as tree edges in the aggregation trees. As the baseline algorithm is essentially a centralized algorithm implemented in a distributed manner, it always finds the minimum q-edge forest in an aggregation network. Next, we present a purely distributed algorithm.

Fragment-based Algorithm. Our distributed algorithm starts with each node being a fragment and iteratively merges them until a forest of q edges is found. Each node has level 0 and the node ID is the fragment's ID. In each iteration, two fragments are combined if they have the same *minimum weight outgoing edge (MWOE)*. MWOE is an edge of minimum weight with two endpoints on two fragments. Each fragment repeatedly performs below two steps viz. *finding MWOE* and *merging fragments via the MWOE* until q edges are found. Initially, all edges start as *basic edges*. Once selected in the q-edge forest, it is called a *tree edge*. If an edge is determined not to be part of the MST, it becomes a *rejected edge*.

Step 1. Finding MWOE. Each level-0 fragment marks its MWOE as a tree edge and sends a message to the node on the other side. The edge chosen by both nodes then merges the two nodes, which becomes a new fragment with level 1.

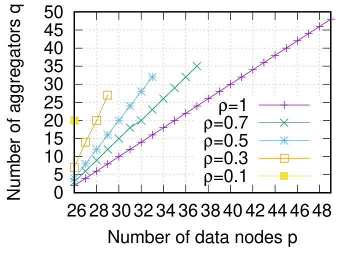
For each non-level-0 fragment to find its MWOE, its leader, the end node of the MWOE with a smaller ID, sends an *initiate* message to the members of the fragment along the *tree edges*. Upon receipt, each node n sends its fragment ID and level along its *basic edges* to node n' on the other end. Then n' compares them with its fragment ID and level and makes the following decisions. (a) If FragmentID(n) = FragmentID(n'), then n and n' belong to the 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' postpones the response until Level(n') $\geq$ Level(n).

Next, all the leaves in the fragment send their MWOE (if there is any) along the tree edge back to its parent. 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 of the fragment gets the MWOEs from all its neighbors and identifies the one with the minimum weight as the MWOE for the entire fragment. Finally, the leader sends a broadcast message to the entire fragment about this new MWOE via the tree edges and starts the fragment merging step described below.

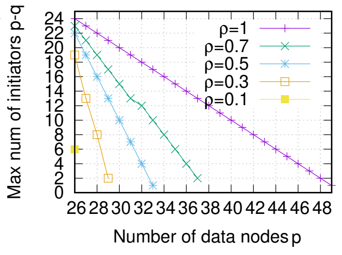
Step 2. Merging fragments via the MWOE. Upon receipt of the MWOE message, the end node of the MWOE within the fragment, say n, becomes the new leader. It marks this MWOE as a tree edge and sends a "request to combine" message to the other end of the MWOE, say n'. Under two scenarios, these two fragments 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 a smaller 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 the "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 along the tree edges. Upon receipt, all the nodes update their parent, children, and fragment identifier (the ID of the new leader).

After each merging, the leader of the newly formed fragment sends a unicast message to a centralized coordinator (node with the smallest ID) about the number of edges in this fragment. Once the number of edges found reaches q, the coordinator sends a broadcast message to the entire network to stop the aggregation tree construction process.

The above distributed algorithm is inspired by a classic distributed MST algorithm (referred to as GHS Algorithm [21, 41]) with the difference that instead of finding an MST of a network, we only need to find one q-edge forest. While the resultant single tree in GHS Algorithm must be an MST, the



(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 ($R=m$).

resultant q-edge forest of multiple trees is not necessarily a minimum q-edge forest. This is because an MWOE found in Distributed DAO^2-U is not necessarily one of the q smallest cycle-less edges. We leave finding a distributed minimum q-edge forest as future work. To find out how close the resulting q-edge forest is to the minimum one, Section VI compares it with the baseline algorithm, as it always finds the minimum q-edge forest.

Stage 3: Data aggregations. Recall that in data aggregation, an initiator of each fragment visits all the nodes in the tree following the longest-path walk defined in Section III-B2. Thus we select the initiator as one of the two leaf nodes on this longest path with the smaller ID. To find the initiator of a tree, we need to run the DBF algorithm to find the shortest path cost between any two leaf nodes in the tree. 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 (i.e., the longest path) and smaller ID knows it is the initiator. It then transmits its data to visit all the nodes in the tree while traversing the edges on this longest path once.

Time and Message Complexity. Table II summarizes the time and message complexity of the three stages of the Distributed DAO^2-U, where its second stage is fragment-based. For data aggregation, the first number in each field is for broadcast; the second is for unicast.

TABLE II
TIME AND MESSAGE COMPLEXITIES IN DISTRIBUTED DAO^2-U.

	Time Complexity	Message Complexity
Aggregation Network	$O(p\|E\|)$	$O(p\|E\|)$
q-edge Forest	$O(p \cdot \log p)$	$O((p+\|E\|)\cdot \log p)$
Data Aggregation	$O(\|V\|)$, $O(\|V\|^2)$	$O(\|V\|)$, $O(\|V\|^2)$

Theorem 6: Distributed DAO^2-U finds an optimal aggregation cost when one initiator is allowed.

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

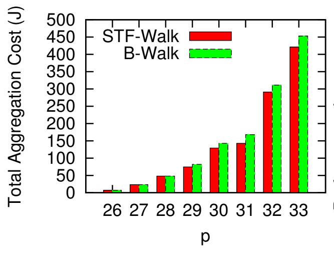
B. Distributed Algorithms for DAO^2

The Distributed DAO^2 is the distributed implementation of the two DAO^2 algorithms viz. approximation (Algo. 2) and Benefit-based (Algo. 3) respectively. It consists of three stages: aggregation network construction, aggregation tree construction, and data aggregation along the aggregation trees. The first stage is the same as in the Distributed DAO^2-U. Below we illustrate the second and third stages for Distributed Approximation and Distributed Benefit, respectively. Initially, each data node in the aggregation network sends a broadcast message to the entire network about its prize.

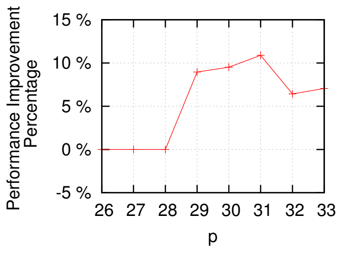
Algo. 2 merges two fragments with a maximum prize-cost ratio iteratively. In its distributed implementation, the leader of each fragment computes the prize-cost ratios with all other fragments and broadcasts the largest value to other fragments. Then the leaders of the two fragments with the maximum prize-cost ratio communicate to update the new leader (smaller ID being the leader) and total collected prizes in the newly formed fragment. In the process, the leader of each newly formed fragment reports the collected prizes to the centralized coordinator. When the total prizes collected reach $\mathcal{Q}$, the coordinator sends a message to the entire network to stop the prize-collecting process. We refer to this implementation as *Distributed Approximation*.

Algo. 3 iteratively adds cycleless edges with maximum benefit and therefore is not as amenable as Algo. 2 for distributed implementation. We thus adopt the baseline distributed implementation for DAO^2-U while considering the prizes. In particular, a selected leader (node with the smallest ID) gathers the topological information of the data aggregation network together with the available prizes at nodes and executes Algo. 3 to find data aggregation trees of the aggregation network. It then broadcasts this result to all the data nodes, each of which then knows whether each incident edge belongs to an aggregation tree. We refer to this implementation as *Distributed Benefit*.

We assume that all the sensor nodes have enough energy to perform the tasks computed by the centralized (i.e., Algo. 1, 2, and 3) and distributed algorithms (i.e., Distributed DAO^2-U, Distributed Approximation, and Distributed Benefit). However, when sensor nodes have limited energy, some might deplete their energy, thus causing network partition in the data aggregation process. Our distributed algorithms can be modified accordingly to address the network partitions. For example, in the Distributed DAO^2-U, if the energy level of a node on the

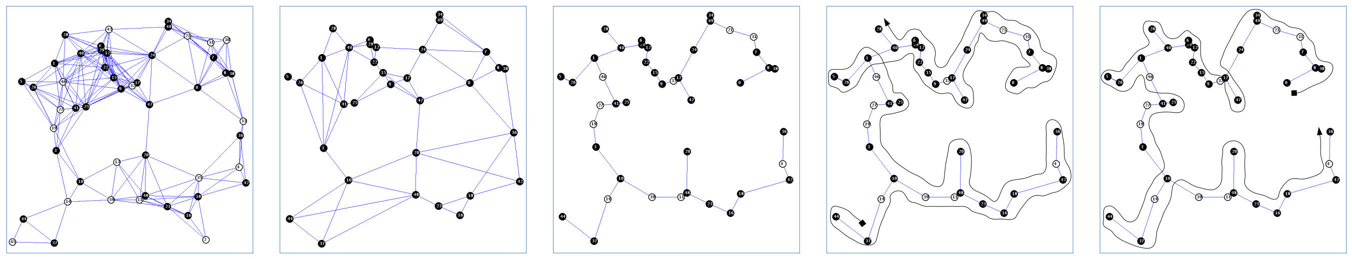


(a) Total aggregation cost.



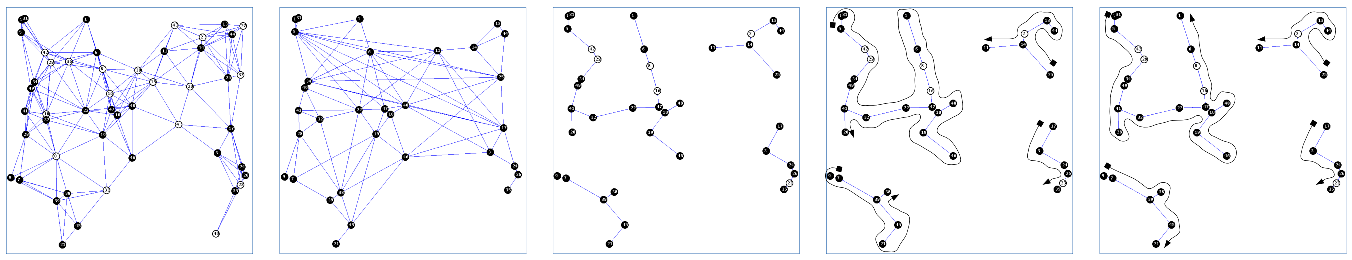
(b) Performance improvement.

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



(a) ICSN 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) ICSN 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 four 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.

longest path of an aggregation tree is below a pre-specified threshold, it will notify the local neighborhood so that a new path can be established for the data aggregation before it is too late. We leave its detailed implementation as future work.

VI. Performance Evaluation

This section investigates and compares all the data aggregation algorithms with the state-of-the-art. Table III lists all the compared algorithms. We write our simulators in Java on a MacBook Pro with a 2 GHz Dual-Core Intel Core i5 processor and 8 GB RAM. 50 or 100 sensors are uniformly distributed in a region of 1000m $\times$ 1000m or 2000m $\times$ 2000m. The transmission ranges of sensor nodes are 250m. The correlation coefficient ρ is given as a fixed constant - How to estimate it has been studied by existing literature [54]. Each data point is averaged over ten runs, and the error bars indicate 95% confidence interval wherever applicable. Below we first evaluate the centralized algorithms for DAO^2-U and DAO^2 in Section VI-A and VI-B respectively, and then their distributed implementation in Section VI-C.

TABLE III
Summary of All Algorithms.

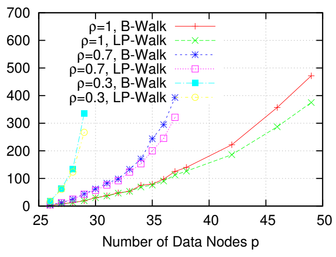
	DAO^2-U	DAO^2	Existing Work
Centralized	B-Walk (Algo. 1), STF- & LP-Walk	Approx. (Algo. 2) Benefit (Algo. 3)	S-Tree
Distributed	Fragment, Baseline	Dist. Approx. Dist. Benefit	Dist. S-Tree

A. Algorithms for DAO^2-U

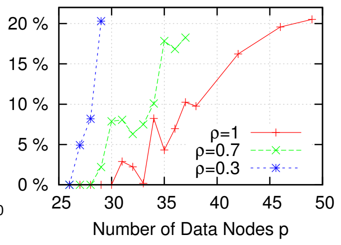
We first consider 50 nodes in a 1000m $\times$ 1000m region. Unless otherwise mentioned, $R = m = 512$MB.

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 the 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. More data nodes mean 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 by setting $\rho = 0.5$, a representative correlation coefficient, and varying p from 26 to 33. Fig. 9(a) shows that when $p \le 28$, 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 the

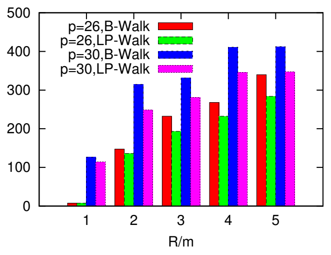


(a) Total aggregation cost (KJ).

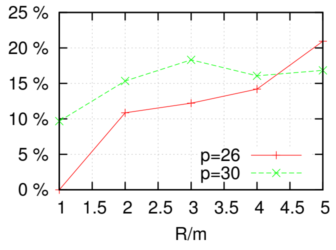


(b) Performance improvement.

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



(a) Total aggregation cost (KJ).



(b) Performance improvement.

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

connected components of the resultant q-edge forests to be all linear. In linear topologies, aggregation occurs by traversing from one end of the linear topology to the other, resulting in the same performances for STF- and B-Walk. However, when p gets larger, STF-Walk performs better than B-Walk, because STF-Walk always traverses the smaller subtree twice while B-Walk could 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 adopt STF-Walk instead of B-Walk but still refer to it as B-Walk.

Comparing B-Walk with LP-Walk Visually. Before comprehensively comparing 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 an ICSN 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.

Multiple Initiators Case. We consider $\rho = 0.5$ and $p = 32$, which has 28 aggregators and allows at most four initiators, as shown in Fig. 11. It shows that B-Walk traverses 19 edges twice, resulting in a total aggregation cost of 255.9J, while LP-Walk traverses nine edges twice, with a total aggregation cost of 203.0J, a 20.7% improvement. Compared to Fig. 10, when more initiators exist, the performance difference between B-Walk and LP-Walk could get smaller. In particular, Fig. 11(d) and (e) show that they find the same aggregation walks for two smaller trees on the right. With more initiators allowed, the resultant q-edge forest consists of more trees of small sizes, each with a "short" longest path. By traversing edges on such short longest paths once, LP-Walk does not save as much as it traverses a big tree with a much longer longest path. Finally, compared to the single initiator case, both B-Walk and LP-Walk incur less energy, as more initiators are utilized to find cost-effective aggregation walks.

Comparing B-Walk with LP-Walk by Varying p and ρ. Next, we compare the aggregation costs in B-Walk and LP-Walk in the 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 fewer aggregators are visited, thus incurring fewer aggregation costs.

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 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 than $\rho = 0.7, 1$. When less data correlation exists, more aggregators are visited, making larger sizes of the resultant q-edge forest and its constituent trees. LP-Walk can thus save more aggregation costs than traversing smaller trees by traversing the longest paths of larger trees once. Given any ρ, when increasing p, the number of allowed initiators $p - q$ decreases (Fig. 8(b)). As such, performance improvement of LP-Walk upon B-Walk increases, as shown in Fig. 12(b).

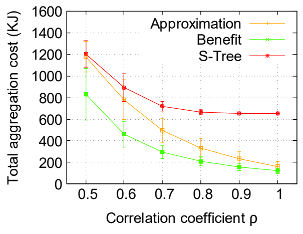
Comparing B-Walk with LP-Walk by Varying R/m. We compare B-Walk with LP-Walk on different R/m. When increasing R/m, the overall storage overflow situation gets more challenging since there is relatively more overflow data than available storage spaces. We choose $\rho = 0.5$ and vary R/m from 1 to 5, 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 resulting q-edge forests get larger. This favors LP-Walk, which travels a large number of edges only once.

B. Algorithms for DAO^2

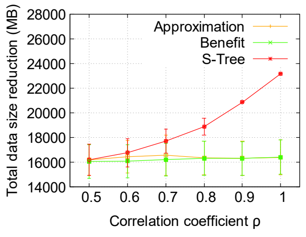
State-of-the-art Data Aggregation Techniques. Existing works of data aggregation in sensor networks mainly use tree-based routing structures that connect the base station and sensor nodes [2, 13, 30, 32, 36, 51]. Data at sensor nodes are transmitted back to the base station along the tree while being aggregated. As existing works consider a single aggregation tree that spans the base station and sensor nodes while there is no base station in DAO^2, to make a fair comparison, we construct one aggregation tree by modifying Algo. 2. In particular, it merges two fragments with the largest prize-cost ratio until the largest fragment collects enough quota of Q. This largest fragment serves as the data aggregation tree for existing work. Then, the node with the 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.

We refer to the above single-tree data aggregation paradigm by existing works as **S-Tree**, the approximation algorithm Algo. 2 as **Approximation**, and the heuristic algorithm Algo. 3 as **Benefit**. We randomly generate 100 sensor nodes (50 data nodes and 50 storage nodes) in a 2000m $\times$ 2000m field with a transmission range being 250m. As we study the general case of DAO^2, we set R_i as a random number in [512MB, 1024MB] and m_j a random number in [256MB, 512MB]. To make the data aggregation feasible (i.e., there is enough data size reduction after aggregation), instead of randomly setting the correlation coefficient ρ_i and then checking its feasibility, we assume all the data nodes have the same correlation coefficient (i.e., $\rho_i = \rho$). We define the *threshold correlation coefficient* as the minimum feasible correlation coefficient of all the data nodes and denote it as ρ_{th}. That is, $\rho_{th} = \frac{\sum_{i \in V_d} R_i - \sum_{j \in V - V_d} m_j}{\sum_{i \in V_d} R_i}$. We then increment ρ for all the data nodes from ρ_{th} to 1 in 0.1 stepwise.

Fig. 14(a) shows that Benefit outperforms Approximation, which outperforms S-Tree in the entire range of ρ. The total aggregation costs of all three algorithms decrease with the increase of ρ. This is because when ρ is small, more aggregators are needed; thus they are more likely to form a few large fragments. Visiting all the aggregators in these large fragments from a few initiators is more costly than visiting more smaller fragments. This shows that the more spatial correlation of the data, the more cost-efficient our data aggregation techniques are. Approximation performs more like S-Tree when ρ is small and more like Benefit when ρ is large. This is because the larger ρ, the larger prizes available at each node. Therefore, the fragment-merging process in Approximation results in one or a few large fragments when ρ is small (like S-Tree) and a single edge is merged in a fragment each time as Benefit does when ρ gets large. Benefit outperforms S-Tree between 31.6% to 71.8% in the entire range of ρ.



(a) Total aggregation cost (KJ).



(b) Total data size reduction.

Fig. 14. Comparing three DAO^2 algorithms.

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

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

Fig. 14(b) shows the total size reduction achieved in all three algorithms (while the average required overflow data size Q is 15,904 MB). It shows that both Approximation and Benefit reduce around 16,000 MB data size for the entire range of ρ, around 3% of more reduction than required. A close look shows that Approximation aggregates more data than Benefit at $\rho = 0.6$ and 0.7, sustaining our initial conjecture that Approximation could yield more prizes (i.e., aggregates more data) than Benefit. In contrast, S-Tree always looks for a single aggregation tree to visit all the aggregators. With the increase of ρ, as fewer aggregators are needed, S-Tree thus needs more nodes to connect the targeted aggregators, dramatically increasing the total size of data reduction.

Table IV shows the number of initiators a and the 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, the number of initiators a increases for both Approximation and Benefit. This is because the fewer aggregators in the sensor field, the more distance among them thus the smaller chance that they are merged into fragments, resulting in more fragments. As each fragment has one initiator, the number of initiators increases with the increase of ρ. Note that S-Tree has only one initiator as it always finds one aggregation tree.

Fig. 15 visually shows the resultant aggregation trees from a typical run of the three algorithms. Benefit has 13 small trees, whereas Approximation has four and S-Tree has one. This demonstrates the more granular effort of Benefit in data aggregation, wherein each initiator only visits its local data nodes for aggregation. In contrast, in Approximation and S-Tree, initiators could travel long distances for aggregation, thus costing more energy.

C. Distributed Algorithms

Finally, we evaluate the performances of our designed distributed algorithms for DAO^2-U and DAO^2-U, respectively. 100 nodes are randomly placed in a 2000m $\times$ 2000m region. The transmission range is 250m.

1) Distributed Algorithms for DAO^2-U: We set $m = R =$ 512MB and compare two distributed algorithms viz. Baseline and Fragment-based. Overhead messages of different sizes are used in all three stages of our distributed algorithms. In Stage 1: Aggregation network construction, the overhead message used in distributed Bellman-Ford is 20 B, and the overhead message used for constructing the aggregation network is 1000 B, as the message includes the shortest paths to all other nodes. There are two distributed algorithms viz. Baseline and Fragment-based in Stage 2: Constructing q-edge Forest. In Baseline, the size of the message selecting a leader is 20 B, and the size of the leader's message informing the computed results is 1000B. In Fragment-based, the overhead messages finding MWOE and merging fragments are 20 B. In Stage 3: Data aggregation, the message finding the initiator of the LP-Walk is 20 B. As the overflow data packet size of 512 MB is very large, in our simulations, it is fragmented into 1000 data packets, each of 512 KB.

We adopt a typical correlation coefficient value $\rho = 0.6$, find the valid range of the number of data nodes p using Equation 2, which is [51, 71], and vary p in this range. Table V shows the corresponding number of initiators a found in both Baseline and Fragment and the number of aggregators q. With the increase of the number of data nodes p, the number of initiators needed for energy-efficient data aggregation in both algorithms increases first and then decreases. When increasing p initially, with the increased overflow data, more aggregators need to be visited to achieve the required data reduction. With the increasing number of aggregators, more connected fragments are formed. As each tree needs one initiator, this results in an increased number of initiators in the initial stage of increasing p. However, with the further increase of p, the number of aggregators increases to the extent that the different trees they belong to begin to merge, resulting in fewer trees and initiators.

Table VI shows the breakup of the energy cost in both Baseline and Fragment. Among the three stages in the distributed algorithms, the first two stages, viz. constructing aggregation network and finding q-edge forest, cost the smallest portion of energy, in the order of a few or tens of Joules, while the third stage, viz. data aggregation costs the most of the energy, in the order of Kilojoules. This shows that in data-intensive ICSNs, most energy is spent on the payload of data aggregation instead of on the overhead cost in aggregation network construction and finding q-edge forest. This demonstrates the energy efficiency of our data preservation system.

2) Distributed Algorithms for DAO^2: We set R_i as a random number in [512MB, 1024MB] and m_j as a random number in [256MB, 512MB]. We consider the distributed algorithms of Approximation (Algo. 2), Benefit (Algo. 3), and S-Tree. Recall that S-Tree combines fragments with the largest prize-cost ratio iteratively like Approximation, except that it won't stop until the largest fragment collects enough quota of Q. Therefore, S-Tree's distributed implementation follows the implementation of Approximation proposed in Section V-B, except that the coordinator records the prizes collected by each fragment. When the largest prize of any fragment reaches Q, it sends a message to the entire network to stop the prize-collecting. Finally, the node with the smallest prize in this fragment is selected as the base station. From this, the 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. Fig. 16(a) shows the distributed S-Tree incurs much more energy cost than distributed Approximation and Benefit. This can be explained by Fig. 16(b), which shows that distributed S-Tree collects more prizes than necessary. Again, as S-Tree only allows a single data aggregation tree, its aggregation cost is much higher than that of the other two distributed algorithms.

TABLE V
THE NUMBER OF INITIATORS a AND THE NUMBER OF AGGREGATORS q W.R.T. THE NUMBER OF DATA NODES p.

p		55	60	65	70	71
q		17	34	50	67	70
a	Baseline	11.3	13.2	9.5	2.5	1
	Fragment	14.8	18.9	14.3	2.8	1

VII. Related Work

Below we review the prior work in the theory and sensor networking communities, respectively.

A. Multiple and Quota Traveling Salesmen Problems

TSP^2 is different from well-known multiple traveling salesmen problem (mTSP) [10] and vehicle routing problem (VRP)

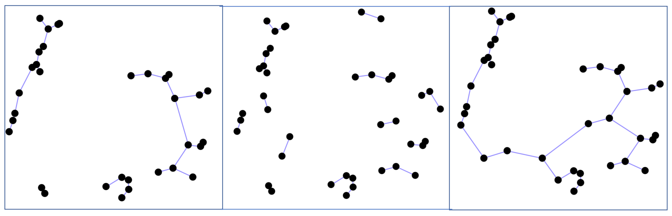


Fig. 15. Aggregation trees in (a) Approximation, (b) Benefit, and (c) S-Tree.

TABLE VI
ENERGY CONSUMPTION OF DISTRIBUTED ALGORITHMS FOR DAO^2-U. STAGE 1: AGGREGATION NETWORK CONSTRUCTION. STAGE 2: FINDING q-EDGE FOREST. STAGE 3: DATA AGGREGATION.

p		55	60	65	71
Stage 1 (J)		55.26	51.42	53.23	55.4
Stage 2	Baseline (J)	2.85	3	3.24	3.42
	Fragment (J)	0.81	1.74	2.85	4.55
Stage 3	Baseline (KJ)	77.79	228.62	684.42	1682.95
	Fragment (KJ)	134.25	300.82	696.48	1674.81

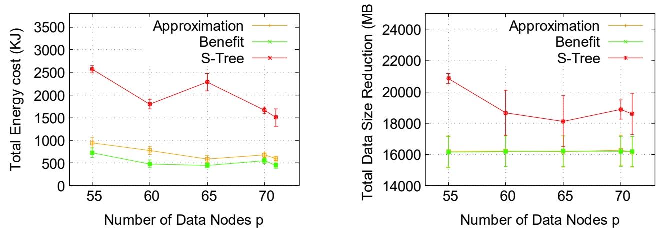
[52]. mTSP determines a set of routes for multiple salesmen who all start from and return to a single city (or a set of cities), such that all the cities are visited exactly once and the total cost of visiting all the cities is minimized. VRP generalizes mTSP by adding more constraints, such as time windows for pickup and delivery, and that vehicles have capacities. In both mTSP and VRP, a prescribed set of cities (or customers) must be visited, and each traveling route must start and end at the prefixed (same or different) depots. However, TSP^2 requires all together q cities to be visited, while a salesman can end at a node different from his starting node. As such, TSP^2 needs to decide how many salesmen are required, where to place them, and how to find each route.

Q-TSP^2 is different from the *prize-collecting TSP* (or TSP/vehicle routing with profit) [4, 8, 19]. 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. The complementary problem of maximizing the collected profit with the budgeted traveling cost is *orienteering problem* [53]. As our work is to aggregate and reduce the size of sensory data by some specified amount while minimizing the energy cost incurred, it bears more resemblance to the prize-collecting TSP. Below we review its related literature.

Bienstock [11] decided on 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. [3] further improved the approximation ratio to $2-\epsilon$ using Lagrangian relaxation. Tang and Wang [50] proposed a local search-based heuristic for a related capacitated prize-collecting TSP. Dell'Amico et al. [17] developed a lagrangian heuristic and obtained an upper bound in the form of a feasible solution.

If no penalty is considered, prize-collecting TSP becomes a quota-TSP problem [5, 6]. Awerbush [6] proposed an $O(\log^2 R)$ approximation algorithm where R is the quota to collect. It is based on an approximation for the k-minimum-spanning-tree problem (k-MST), finding a tree of the least weight that spans exactly k vertices on a graph. Ausiello et al. [5] studied the online version of the problem.

However, the above prize-collecting or quota TSP research assumes only one traveling salesman. If multiple traveling salesmen are allowed, the prize-collecting mTSP has been studied under the name of multi-vehicle routing with profit, to which Chapter 9 in [19] gave a comprehensive survey of the current work. All of them, however, only focus on routing while fixing the number of vehicles and the starting and ending depots of the vehicles. DAO^2 significantly differs from existing research for two reasons. First, not only does DAO^2 needs to decide the number of salesmen but also to determine the starting and ending node of each salesman; that is, it studies both placements and routing of traveling salesmen in the same problem space. Our work is the first one to study multiple TSP prize-collecting problems considering both placement and routing, wherein the number of salesmen, the starting and ending node of each salesman, and the route of each salesman must all be decided simultaneously. Second, none of the existing TSP or mTSP problems consider that traveling salesmen, when dispatched from different nodes, could have different traveling modes and incur a different cost per unit distance traveled, which uniquely arises and are identified in DAO^2. How the cost-and-time tradeoff among various transportation means (e.g., cars, railways, and air) impacts the solutions to various TSP problems remains largely unexplored.



(a) Total aggregation cost (KJ). (b) Total data size reduction.

Fig. 16. Distributed algorithm performance comparison for DAO^2.

B. Data Aggregation in Sensor Networks

In the sensor network community, extensive research focuses on disconnection-tolerant operations without the base station. Some system research was conducted to design cooperative distributed storage systems and to improve the utilization of the network's data storage capacity [37, 38]. Other research instead took an algorithmic approach by focusing on the optimality of the solutions [25, 49, 58]. However, all the above works assumed that enough storage space is available to store the overflow data, thus not addressing the overall storage overflow problem.

There is a vast amount of literature on data aggregation in sensor networks [2, 13, 30, 32, 36, 51]. Tree-based routing structures were often proposed to either maximize network lifetime (the time until the first node depletes its energy) [36], minimize total energy consumption or communication cost [30, 32], or reduce the delay of data gathering [2]. Recently, Chen et al. [13] considered the duty-cycle sensor networks and designed two distributed data aggregation algorithms where an 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 to maximize the network lifetime [45, 51]. However, data aggregation in DAO^2 significantly differs from existing data aggregation techniques in both goals and techniques. First, existing data aggregation reduces the number of transmissions by combining data from different sensors en route to the base station to save energy. However, the goal of data aggregation in DAO^2 is to aggregate the overflow data so that they can fit into storage available, thus preventing data loss caused by overall storage overflow in the ICSN. 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 unavailable, those routing structures are no longer suitable. Instead, DAO^2 introduces a minimum q-edge forest, a routing structure that serves as the building block of our techniques.

ICSNs differ from delay-tolerant sensor networks (DTSN) [33]. In DTSNs, mobile nodes are intermittently connected due to their mobility and low density, and data is opportunistically forwarded to destination nodes. In ICSNs, however, all the static sensors are connected while disconnected from the base station, and data is uploaded to the base station only when uploading opportunities such as data mules become available.

VIII. Conclusion and Future Work

We introduced an algorithmic framework called DAO^2, which tackles the overall storage overflow problem in emerging sensor network applications. Our work has two theoretical significance. First, in studying DAO^2, we uncovered two new graph-theoretic problems viz. TSP^2 and Q-TSP^2. In sharp contrast to classic TSP and its variants that mainly focus on the routing of salesmen, TSP^2 and Q-TSP^2 find both the placement and the routing of the traveling salesmen. We designed a suite of energy-efficient optimal, approximation, heuristic, and distributed data aggregation algorithms to solve these two problems. Second, the two building blocks of our framework viz. aggregation network and minimum q-edge forest are two novel graph structures not formally identified and studied in any existing research. The minimum q-edge forest generalizes the minimum spanning tree, one of the most fundamental graph data structures. As such, one of our approximation algorithms (Algo. 1) generalizes the well-known Kruskal's MST algorithm. Of these theoretical roots, the techniques proposed in this paper could be applied to any applications wherein data correlation and resource constraints coexist, including Internet of Things applications, data centers, and big data analytics.

One limitation of our work is that the data nodes and their overflow data are known beforehand. In a real-time scenario wherein data nodes emerge dynamically, the $extDAO^2$ can still be solved by periodically executing our algorithms whenever new data nodes arise. Second, when sensor nodes have a limited amount of energy, some could deplete their energy, causing network partition in the data aggregation process. In the future, we will design more fault-tolerant distributed data aggregation algorithms to tackle such problems. Different spatial correlation models (lossy or lossless) could lead to different aggregation mechanisms. Our model is lossy in that aggregator does not have enough information to inform other data nodes to aggregate. In the future, we will consider a lossless aggregation wherein a sensor node can be both an initiator and aggregator and design new techniques to solve the overall storage overflow problem. Finally, we treat data aggregation and offloading as two separate stages to solve each problem nicely. Although both stages are solved optimally or with approximation, their combined solution is not necessarily optimal or achieves the same approximation for the data aggregation and offloading problem. We will integrate these two stages to achieve an energy-efficient solution for the overall storage overflow problem.

Acknowledgment

This work was supported in part by NSF Grants CNS-1419952 and CNS-2131309.

References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

[9] 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.

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

[11] 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.

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

[13] 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.

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

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

[16] 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.

[17] 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.

[18] S. Edwards, T. Murray, T. O'Farrell, I. C. Rutt, P. Loskot, I. Martin, N. Selmes, R. Aspey, T. James, S. L. Bevan, and T. Baugé. A High-Resolution Sensor Network for Monitoring Glacier Dynamics. *IEEE SENSORS JOURNAL*, 14(11):3926–3931, 2013.

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

[20] 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.

[21] 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.

[22] R. Ghaffarivardavagh, S. S. Afzal, O. Rodriguez, and F. Adib. Ultra-wideband underwater backscatter via piezoelectric metamaterials. In *Proc. of the ACM SIGCOMM*, 2020.

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

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

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

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

[27] H. Huang, A. V. Savkin, M. Ding, and C. Huang. Mobile robots in wireless sensor networks: A survey on tasks. *Computer Networks*, 148:1–19, 2019.

[28] J. Jang and F. Adib. Underwater backscatter networking. In *Proc. of the ACM SIGCOMM*, 2019.

[29] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta Informatica*, 15:141 – 145, 1981.

[30] 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.

[31] 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):62:1–62:30, 2014.

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

[33] 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.

[34] W. Liang, X. Ren, X. Jia, and X. Xu. Monitoring quality maximization through fair rate allocation in harvesting sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 24(9):1827–1840, 2013.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

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

[41] 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.

[42] D. E. Phillips, M. Moazzami, G. Xing, and J. M. Lees. A sensor network for real-time volcano tomography: System design and deployment. In *Proc. of IEEE ICCCN 2017*.

[43] M. Rahmati and D. Pompili. Uwsvc: Scalable video coding transmission for in-network underwater imagery analysis. In *Proc. of IEEE MASS 2019*.

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

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

[46] 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.

[47] Rui Tan, Guoliang Xin, Jinzhu Chen, Wen-Zhan Song, and Renjie Huang. Fusion-based volcanic earthquake detection and timing in wireless sensor networks. *ACM Transaction on Sensor Networks (ACM TOSN)*, 9, 2013.

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

[49] 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.

[50] 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.

[51] 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.

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

[53] 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.

[54] 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.

[55] 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.

[56] A. Wichmann, T. Korkmaz, and A. S. Tosun. Robot control strategies for task allocation with connectivity constraints in wireless sensor and robot networks. *IEEE Transactions on Mobile Computing*, 17(6):1429–1441, 2018.

[57] S. Xie, G. X. Lee, K. S. Low, and E. Gunawan. Wireless sensor network for satellite applications: A survey and case study. *Unmanned Systems*, 02(03):261 – 277, 2013.

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

[59] H. Yedidsion, A. Banik, P. Carmi, M. J. Katz, and M. Segal. Efficient data retrieval in faulty sensor networks using a mobile mule. In *Proc. of WiOpt 2017*.

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

[61] 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.