The following publication P. Guo, X. Liu, J. Cao and S. Tang, "Lossless In-Network Processing and Its Routing Design in Wireless Sensor Networks," in IEEE Transactions on Wireless Communications, vol. 16, no. 10, pp. 6528-6542, Oct. 2017 is available at https://doi.org/10.1109/TWC.2017.2724516.

Lossless In-network Processing and Its Routing Design in Wireless Sensor Networks

Peng Guo, Xuefeng Liu, Jiannong Cao and Shaojie Tang

Abstract—In many domain-specific monitoring applications of wireless sensor networks (WSNs), such as structural health monitoring, volcano tomography and machine diagnosis, the raw data in WSNs are required to be losslessly gathered to the sink where a specialized centralized algorithm is then executed to extract some global features or model parameters. To reduce the large raw data transmission, in-network processing is usually employed. However, different from most existing in-network processing works that pre-assume some common computation/aggregation functions, in-network processing of a given centralized algorithm requires to exactly partition the algorithm first and then appropriately assign the partitioned computations into WSNs. We call it as lossless in-network processing, which has not gotten much studied. Lossless in-network processing raises two questions: 1) what pattern should a centralized algorithm be partitioned into so that the partitioned computations can be flexibly assigned into a WSN with arbitrary topology? and 2) for each partition pattern how to design efficient routing for the resource-limited sensor nodes? These two questions can be called as topology-constrained computation partition problem and computation-constrained routing design problem, respectively. In this paper, we first introduce some general patterns on the topology-constrained computation partition. Then, with the computation constraints in the patterns, we present a series of novel routing schemes customized for different cases of computation results. The work in this paper can also serve as a guideline for distributed computing of big data where the data spreads in a large network.

Index Terms—Wireless Sensor Network (WSN), In-network processing, Matrix computation, Routing scheme

1 Introduction

Recently, we are seeing that wireless sensor networks (WSNs) are extending their applications into some domain-specific areas, such as structural health monitoring (SHM) [1], volcano tomography [2], state estimation in smart grid [3], and fault diagnosis of machines [4]. In these domain-specific applications of WSNs, usually large amount of sensor nodes are deployed and **cooperatively monitor a common huge object** (e.g., structure, volcano, smart grid and large machine) instead of individual events in the network. After the sink node gathers the raw data sampled at different positions of the object, a specialized centralized algorithm is then executed to extract some global features or estimate model parameters of the object. We call such kind of monitoring applications as global collaboration monitoring (GCM).

To obtain the global feature or model parameter, GCM usually requires each node to sample sufficient raw data. For example, to estimate the mode shape of a structure in structural health monitoring, the sampled local vibration data at each sensor node can be of a size about several thousands of bytes [5]. To reduce the large amount raw data transmission in WSNs, in-network processing is an efficient way. With in-network processing, only the important information or intermediate computation data rather than the raw data is transmitted. Hence, the transmission load in WSNs can be significantly reduced.

However, implementing in-network processing in WSNs for GCM is not easy. Due to the global collaboration on common object monitoring, the algorithm for GCM usually works with data-level collaboration, which is much different from the feature-level or decision-level fusion employed in conventional WSNs applications. Fig. 1 shows the difference among the three collaboration modes. For the feature-level or decision-level fusion, each sensor can process its own data independently to extract feature or make decision without exchanging information with others. Therefore, the in-network processing of these kinds of fusion tasks is usually simple (e.g., MAX, SUM or top-k) and typically limited to spatial or temporal aggregation [6], or directly assumes a general aggregation function (e.g., constant size of output) [13]. However, in WSNs for GCM, the in-network processing of data-level collaboration should not simply assume a common aggregation (or feature extraction) function in advance, because processing the local aggregation results (or features) instead of the raw data cannot achieve the same accuracy with that of the processing on the raw data [14], or even have a far deviated output [15]. Typical data-level collaboration based GCM applications are SHM [1], volcano tomography [2], and fault diagnosis of machines [4]. In these applications, usually a dedicate domain-specific centralized algorithm is designated, and a WSN needs to exactly (or losslessly) distribute the centralized algorithm into sensor nodes for in-network processing, so as to achieve the same quality as that of the centralized GCM system (which is so-called

P. Guo (e-mail: guopeng@mail.hust.edu.cn) and X. Liu are with Huazhong University of Science and Technology, China. E-mail: {guopeng, lxfeng0527}@hust.edu.cn
J. Cao is with Hong Kong Polytechnic University, Hong Kong, China. E-mail: csjcao@comp.polyu.edu.hk.
S. Tang is with University of Texas at Dallas, Richardson, TX 75080. E-mail: shaojie.tang@utdallas.edu.

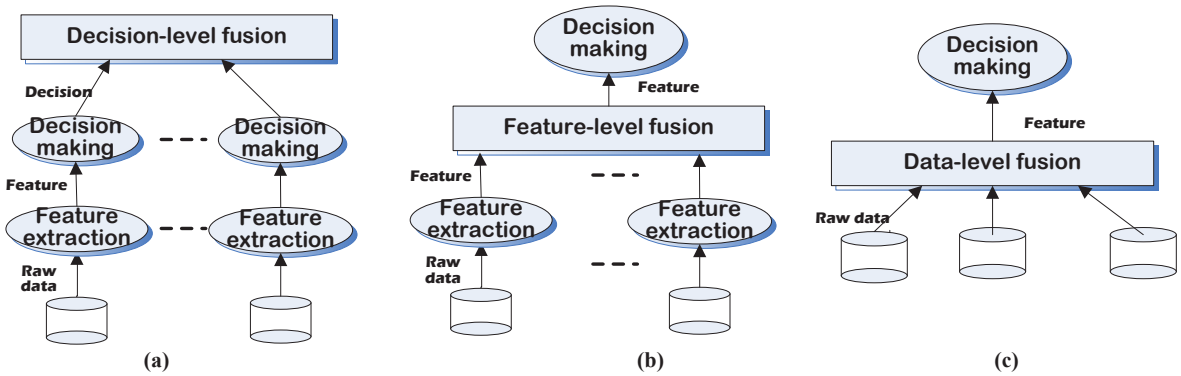


Fig. 1 Algorithms with collaboration (a) at decision level (b) at feature level (c) at data level.

lossless in-network processing in this paper).

Unfortunately, it is not easy to exactly partition the domain-specific GCM algorithms into distributed in-network processing functions, as many of these algorithms contain matrix computations, such as eigen decomposition (ED) or singular value decomposition (SVD) for feature extraction in SHM [1], and matrix inversion in least square estimation (LSE) for model parameter estimation in volcano tomography [2]. For each step in these matrix computations, almost all raw data are involved. However, to in-network process the computation, it is expected to partition the computation into a series of distributed functions each of which requires only part of the raw data. In particular, each distributed function assigned to a sensor node needs only the raw data of the node as well as the raw data from the node's descendant designated by the routing scheme. Considering the arbitrary topology of the WSN in practice, finding appropriate partition of a centralized algorithm for any WSN is not a trivial.

In this paper, we separate the problem of lossless in-network processing into two parts: 1) finding the general computation partition patterns that can be independent of (or say universal to) the network topology; 2) designing the computation-constrained routing for each partition pattern. The objective of the routing design is to minimize the total transmission cost in WSNs, with the constraints of the computation partition patterns.

Contributions of the paper are summarized as follows.

- We introduce some general patterns on lossless computation partition, with which the partitioned computations can be flexibly assigned into a WSN with arbitrary topology.
- For each computation partition pattern, we list several typical rules on the relationship between raw data's size and the size of computation results (i.e., the intermediate data), which widely exist in practical GCM algorithms. Though finding the optimal routing with these rules is generally NP-hard, we find this kind of routing problem can come down to an appropriate combination of the classic routing strategies (e.g., MST, SPT, TSP), with an insight of routing for the above data-size rules.
- Based on the above hint, we design a series of computation-constrained routing schemes, respectively for each rule of the data sizes in each partition pattern. Extensive simulation results show the efficiency of the schemes.

Note that, the work in the paper, either the computation partition patterns or the routing ideas, is not limited to lossless in-network processing in WSN, but can also serve as a guideline for distributed computing of big data where the "raw data" spreads in a large network.

The remainder of the paper is organized as follows. In Section 2, we review the related works. Section 3 presents the general lossless computation partition patterns. Section 4 introduces the proposed computation-constrained routing schemes for different cases of lossless in-network processing. Simulations are described in Section 5, followed by the conclusions in Section 6.

2 Related Work

Though lots of works on in-network processing have been done for WSNs [7] in last decade, most of them work at decision level or feature level. Sometimes they are rather called as in-network aggregation, as they usually pre-assumed a common aggregation function, such as averaging the raw data, finding the maximum or top-K of the raw data, or just a general compression formation [8] [9]. The pre-assumed aggregation functions usually ignore the types of input data. Thus, the sizes of aggregation/processing results simply follow a single rule. However, this paper addresses the in-network processing of data-level collaboration algorithm, which needs to partition the algorithm into several functions with considering different types of input data. Hence, the size of processing result at different sensor nodes may follow different rules, which makes the routing design much different from existing routing works for data aggregation [10]–[13].

Nevertheless, there are still some works handling the in-network processing of data-level collaboration. Aiming at least square estimation (LSE), the classic matrix-

involved computation, some works study lossless in-network processing of LSE, i.e., distributed LSE, in WSNs [16]–[18]. However, the efficient assignment of LSE into WSNs is not addressed. In other word, these works do not further study the efficient routing design for the distributed LSE.

Another classic data-level collaboration computation is ED or SVD, which helps to extract the object's features from the raw data. In [21], the authors study the networked computation of SVD for structural health monitoring in WSN, with designing the optimal routing tree for in-network processing of SVD computation. However, they suppose that the SVD computation is performed upon only the raw data at some selected nodes (i.e., cluster headers). Once the raw data is processed into the SVD results, the results are directly delivered to the sink without being further processed. Therefore, although the routing structure in [21] is based on a tree, the computation structure is simply based on clustering. In other word, it is actually a conventional routing scheme for clustered WSNs, with given certain computation task for only cluster headers.

However, we know that the SVD results actually can further be iteratively processed and updated with the raw data at each node [22]. In our previous work [1], we take one step on the routing design for iterative computing SVD within WSNs. However, as the paper mainly addresses partitioning a certain SVD-contained centralized algorithm, the general computation-constrained routing in the paper does not get much studied. In this paper, we give an extensive study on the routing for lossless in-network processing, with consideration a series of typical computation cases in practical algorithms. The work in the paper is not only suitable to the routing in WSNs but also of some significance to the distributed computing of big data in the Internet.

It is noted that, parallel and distributed computing (PDC) [19], which has been heavily researched in the last forty years, seems to address the same problem as ours. In fact, PDC is intrinsically different mainly in following two aspects. Firstly, conventional PDC usually focuses on how to utilize the computation power of a given number of computation utilities to accomplish a computation task with minimum time, while in-network processing a centralized algorithm in WSN for GCM targets at minimizing the communication cost. Secondly, the data used in computation tasks with PDC are NOT generated from the computational entities and initially can be freely assigned to different entities (e.g., by assuming a common memory) according to how the computational task is decomposed. However, when designing lossless in-network processing of the centralized algorithms, it is highly desirable that each computation entity (i.e. a sensor node), when implements its sub-task, uses only its own data or data from its neighbors.

3 Computation Partition

In a WSN employing lossless in-network processing, each sensor node processes its raw data and the received data according to its computation assigned, and then forward the processing result to its parent node designated by the routing. The processing result usually will be further processed at the parent node along with other data (either raw data or other processing result). Finally, the processing result will arrive at the sink node, and it is desired to be the same as that of the centralized algorithm calculated with all the raw data.

Since the processing results at sensor nodes generally can be further processed in in-network processing, there is a hint that in-network processing of a given centralized algorithm requires the algorithm can be partitioned into some incremental-like computation patterns. In this section, we give a detailed discussion of the computation partition patterns.

3.1 Expected properties of partitioned functions

Generally, a lossless in-network processing function at a sensor node, i.e., the computation partitioned and assigned to the node, can be expressed as $f(\{\star\})$, where $\star$ can be either raw data x sampled by sensor nodes or intermediate data s which is the processing result of another in-network processing function.

We expect to partition the computation of a given centralized algorithm into a series of $f(\{\star\})$ functions nested together, i.e., the output of one $f(\{\star\})$ function is one of the input of another $f(\{\star\})$ function. To assign the nested $f(\{\star\})$ functions to a WSN, it is desired that the input data of each $f(\{\star\})$ function assigned to a sensor node comes from the nearby nodes. However, due to the arbitrary topology of WSNs, the sensor nodes in the WSN may not spread around as required by the nested $f(\{\star\})$ functions. Hence, it is needed to carefully design the nested $f(\{\star\})$ function so that the assignment of the functions can be universal to any WSN. To this end, we expect the $f(\{\star\})$ function to have following properties:

- *Property A*: $\{\star\}$ in $f(\{\star\})$ can have any number of elements. In other word, any node in a WSN can execute $f(\{\star\})$ no matter how many neighbors it has.
- *Property B*: The formation of $f(\{\star\})$ can be independent of the content of the data. In other word, the formation of $f(\{\star\})$ keeps unchanging no matter which node the data in set $\{\star\}$ comes from.
- *Property C*: $f(\{\star\})$ can be executed iteratively or recursively with new input data. In other word, the intermediate data can always be further processed by $f(\{\star\})$ with other intermediate data or raw data. This property allows that any sensor node can perform $f(\{\star\})$ as long as an in-network processing path passes by it.

It can be seen, the above three properties make the assignment of the nested functions free from the net-

work's density, nodes' individuation and routing path, respectively, and hence free from the network topology. Note that, these properties are not necessary condition, but a guideline that we proposed for computation partition. We are seeing that many classic matrix computations (such LSE, SVD) have successfully been partitioned into functions with the properties [17], [24]. Sometimes, property A may not hold, as some function $f(\{\star\})$ may require at least a certain number of input data [1]. To this end, a clustering algorithm can be performed in advance, so that $f(\{\star\})$ can be conducted with the unit of cluster which has sufficient input data.

In addition, $f(\{\star\})$ is also expected to have low complexity, so that it can be performed by sensor nodes. Generally, this property is easy to achieve, as each $f(\{\star\})$ is performed with input of only a small portion of the data, and the MCU in sensor nodes today is becoming more powerful and cheaper (e.g., the price of low-power MCU $STM32F103$ with frequency $70MHz$ and RAM $48KB$ is less than 1$).

3.2 Types of processing functions

In view of input data type (either raw data x or intermediate data s), we divide the $f(\{\star\})$ functions into three categories: $f_1(\{x\})$, $f_2(\{x\},s)$ and $f_3(\{s\})$, as shown in Fig. 2 (a). All the input of $f_1(\{x\})$ must be raw data. In other word, $f_1(\{x\})$ processes only the raw data x. $f_2(\{x\},s)$ can process one intermediate data s with some raw data $\{x\}$. $f_3(\{s\})$ can process multiple intermediate data $\{s\}$ and "merge" them to be one s. Generally, $f_3(\{s\})$ requires more computation cost.

A sensor node may be assigned with one or several functions above, according to the routing scheme. For example, if multiple x and one s are routed to a sensor node, to process these data into one s, the node needs to be assigned with either one $f_1(\{x\})$ and one $f_3(\{s\})$ (nested as $f_3(s,f_1(\{x\}))$, i.e., the output of $f_1(\{x\})$ is an input of $f_3(\{s\})$), or just one $f_2(\{x\},s)$. However, if multiple s are routed to the node, the node must to be assigned with $f_3(\{s\})$.

For an intermediate node, it has at least one s received from others and one x sampled by itself. Generally, we assign the node with one $f_2(\{x\},s)$ first, and then with one $f_3(\{s\})$ if the node has multiple s. As for a leaf node, sometimes it may be assigned with $f_1(\{x\})$, while in more cases it cannot be assigned any function, as $f_1(\{x\})$ function usually requires more than one input raw data for computation. Under this circumstance, the leaf node has to deliver its raw data to a cluster leader for executing $f_1(\{x\})$.

3.3 Computation partition patterns

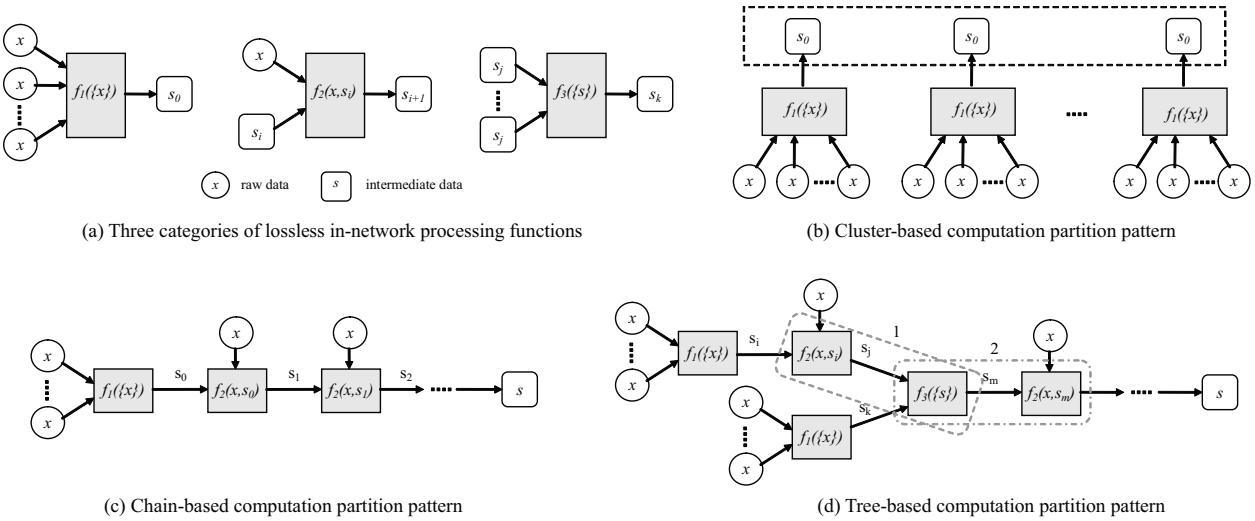
We expect that a centralized computation algorithm can be partitioned into the above three categories of functions nested together. However, it may not be feasible for some centralized algorithms. For example, the distributed least square estimation D-LSE algorithm in [17] does not contain $f_3(\{s\})$, which means two s data cannot be in-network processed into one s. Under this circumstance, the intermediate data s has to always be in-network processed with raw data x by $f_2(\{x\},s)$.

Generally, for any centralized computation algorithm, there are three cases of computation partition for lossless in-network processing, in view of the categories of functions partitioned: i) there is only $f_1(\{x\})$ function in the computation partition; ii) there are $f_1(\{x\})$ and $f_2(\{x\},s)$, but no $f_3(\{s\})$ in the computation partition (e.g., D-LSE algorithm [17]); and iii) all the three categories of functions exist in the computation partition (e.g., SVD in ERA algorithm [1]). The three cases have different constraints on the function nesting structures. We discuss them as follows.

- For the first case, as there is only one $f_1(\{x\})$ function, the processed result of a sensor node can not be furthered processed, which means the centralized algorithm actually can not be partitioned. For this case, sometimes people may directly assign the centralized algorithm to some cluster leader nodes in WSNs, each of which performs the algorithm with partial raw data, as shown in Fig. 2 (b). This idea is similar to "divide-and-conquer", which however does not guarantee the same result of the centralized computation with all raw data.
- For the second case, as there is no $f_3(\{s\})$, i.e., any two intermediate data s can not be processed together, the intermediate data s should always be processed with raw data x by $f_2(\{x\},s)$. Hence, all the $f_2(\{x\},s)$ functions must be nested one by one, thus forming a chain-like nesting structure, as shown in Fig. 2 (c). It can be seen, with the chain-like nesting structure, the centralized algorithm is actually calculated in a recursive way.
- For the third case, as any two functions' output can be the input of a third function $f_3(\{s\})$, there is little limitation on the function nesting structure. Hence, the functions can be nested like an aggregation tree, as shown in Fig. 2(d).

We call these nesting structures as computation partition patterns. Note that, these patterns represent the structure of the computations partitioned, instead of the structure of routes. More specifically, each vertex in Fig. 2 does not refer to one sensor node's computation or data. As analyzed above, a sensor node may be assigned with multiple functions. The functions need to be nested according to the patterns in Fig. 2. For example, if a sensor node has two s and one x, the node will be assigned with one $f_2(\{x\},s)$ and one $f_3(\{s\})$ which are nested as either the structure shown in the dotted box 1 or that shown in dotted box 2 in Fig. 2(d).

Different computation partition patterns impose different constraints on the routing design. In addition, the computation ability of sensor nodes and the variation of data size during the processing also much affect the routing design. In next section, we discuss the routing



(a) Three categories of lossless in-network processing functions

(b) Cluster-based computation partition pattern

(c) Chain-based computation partition pattern

(d) Tree-based computation partition pattern

Fig. 2 Lossless in-network processing functions and some general partition patterns.

design constrained by these computation issues.

4 Computation-constrained routing

In this section, we show how to apply the above general computation patterns into WSNs with arbitrary topology. More specifically, we discuss the efficient routing design under constraints of these computation patterns. The term of "efficient" here refers to the total transmission cost in WSNs with the routing.

4.1 Motivations for computation-constrained routing design

Finding the optimal routing design for lossless in-network processing is usually NP-hard (will be explained later). However, we notice that there are usually some certain rules on the data sizes of processing results in practical engineering algorithms, such as the constant size of processing result for the distributed LSE [17], the linear size increment of processing result for the distributed SVD [1]. These typical rules give us hint on designing the efficient routing. For example, it is well known that: i) for constant size of data transmission at each node in WSNs, the optimal routing is the minimum spanning tree (MST) or Hamilton path; ii) for additive size of data transmission (i.e., the size of processing result is the sum of the input data sizes) at each node, the optimal routing is the shortest path tree (SPT).

However, designing efficient routing for lossless in-network processing is not a simple application of above classic routing strategies. Attributing to the computation involved, there are two more specialties in the routing design for lossless in-network processing, besides the computation constraints.

- Firstly, the data transmitted in the routing has two different rules in the data size. One is the raw data whose size is generally constant, and another is the intermediate data whose size usually follows a typical and uniform rule (e.g., constant or linear increment). These double rules contained in the routing imply a mixture of the classic routing strategies.
- Secondly, the linear increment rule of the intermediate data size can also be regarded as a mixture of two routing strategies. One is for constant size of data transmission corresponding to the constant part of the intermediate data, and another is for additive size of data transmission corresponding to the increased part of the intermediate data (which is generally the sum of the increased part of input intermediate data). Note that, we should not really partition the intermediate data into these two parts for separate transmissions.

With the analysis above, we get motivation that: *the optimal routing design for lossless in-network processing of many practical engineering algorithms, actually can come down to an appropriate combination of the classic routing strategies*. Based on this motivation, in this section we show how to appropriately combine and modify the classic routing strategies to be efficient routing for lossless in-network processing. In particular, as this special routing design highly depends on the rules of data sizes, we discuss the routing design for different cases of data sizes and summarize the corresponding proposed routing algorithms with Table 1, where $|\star|$ denotes the size of $\star$, and $|s|++$ denotes the case that $|s|$ increases with the processing.

4.2 Routing for chain-based computation partition pattern

In chain-based computation, as the intermediate data s should always be processed with raw data x, the network must not initiate more than one intermediate data s. Hence, the in-network processing has to follow a

Table 1 The proposed computation-constrained routing algorithms

Routing for chain-based partition pattern		
	$\|s\| < \|x\|$	$\|s\| \geq \|x\|$
$\|s\| = c$	S-TSP vs CC-TSP	DS-TSP vs CC-TSP
$\|s\| + +$	SP-TSP vs DSP-TSP	

Routing for tree-based partition pattern		
	$\|s\| < \|x\|$	$\|s\| \geq \|x\|$
$\|s\| = c$	LD-MST vs CC-MST	DS-MST vs CC-MST
$\|s\| + +$	DS-SLT vs CC-SPT	

single path to the sink in the WSN, and nodes outside of the path need to deliver their raw data to the nodes on the path for processing. We call the single path on which nodes execute the chain-based computation as *in-network processing path*.

Finding the optimal *in-network processing path* to minimize the total transmission cost in WSNs is a new problem, and can be easily proved to be NP-complete by reducing the travelling salesman problem (TSP) to the problem (with assuming $|s| \rightarrow |x|$). Designing heuristic algorithms for this new problem much depends on the relationship between $|s|$ and $|x|$. We discuss the routing design for different cases as follows.

4.2.1 Case 1: $|s|$ is constant and $|s| < |x|$

Case analysis: In many recursive computation applications where the chain-based partition pattern can be applied, the intermediate data s is usually about the model parameter whose size is smaller than that of the raw data, i.e., $|s| < |x|$. During the recursive computation, the parameter will be updated with each added raw data, while the parameter's size keeps constant. Hence, to save transmission cost of the in-network processing, we expect as many as possible sensor nodes to join the recursive computation, so as to transmit the intermediate data instead of their raw data. Thus, the optimal *in-network processing path* is a Hamilton path from one sensor node to the sink, passing by all other sensor nodes. However, Hamilton path does not always exist in a given WSN, and finding a Hamilton path in the WSN is NP-complete.

We design a semi-TSP routing scheme (named as S-TSP) to find an appropriate *in-network processing path* in a given WSN G. The basic idea of S-TSP is as follows:

- Construct a complete graph G' based on G, by regarding the shortest path between each pair of non-neighboring nodes in G as their edge in G'. According to graph theory, there must exist Hamilton path in G'.
- Employ a TSP greedy algorithm in G' to get a Hamilton path. The path may pass some nodes multiple times in G, and as a result contains some circles on it, as shown in Fig. 3 (a). We call this path in G as semi-Hamilton path, and regard it as the initial *in-network processing path*.
- Update the *in-network processing path* by appropriately removing some nodes from the circles on the *in-network processing path*, if this operation can reduce the total transmission cost in G. The solid line in Fig. 3 (b) is the final *in-network processing path* (which still passes some nodes double times), and the dashed line is the path of raw data delivery.

In addition, we can also directly employ the TSP greedy algorithm on G. Thus, a single but very long path passing part of the nodes in G can be obtained. With regarding this path as the *in-network processing path*, sensor nodes outside of the path will send their raw data, along the shortest routes, to the nearest nodes on the path for processing. We name this routing scheme as CC-TSP (computation-constrained TSP) scheme. Fig. 3 (c) shows the result of CC-TSP in the same network G.

4.2.2 Case 2: $|s|$ is constant and $|s| \geq |x|$

Case analysis: There is also some recursive computation application where $|s| \geq |x|$. For instance, in distributed least square estimation in [17], the size of intermediate data always equals to (or a little larger than) that of raw data. For this case, it is no need to expect the *in-network processing path* to pass each node in G, as sending raw data instead of the intermediated data may have lower cost. However, to avoid long-distance raw data transmission, we still expect the *in-network processing path* detour its way to pass by the *"near"* area of most nodes, so as to let the nodes send their raw data to the path with a short distance.

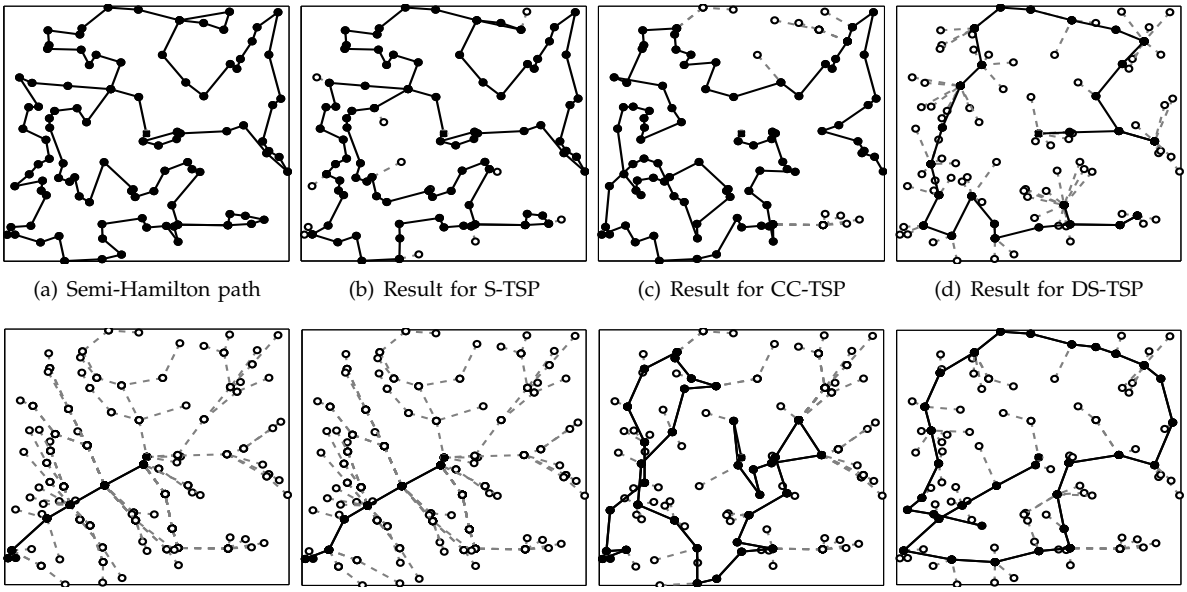
We design a dominating set (DS) plus TSP based routing scheme (named as DS-TSP) for this case. The basic idea is as follows.

- Find a minimum dominating set (MDS) in G.
- Find a Hamilton path passing all the dominant nodes to the sink. The Hamilton path obtained is the *in-network processing path*, and all other nodes outside of path send their raw data to the nearest nodes on the path (i.e., their corresponding dominant nodes).

Fig. 3 (d) shows the result of DS-TSP in the above network. Note that, in DS-TSP we can even use a generalized definition of dominating set. Instead of only dominating one-hop neighboring nodes in traditional DS, we regard the dominating range (i.e., the upper bound of hop counts between dominated node and dominating node) to be $\lfloor |s|/|x| \rfloor$.

4.2.3 Case 3: $|s|$ increases with the processing

Case analysis: For the recursive computation of another well-known matrix computation, i.e., SVD [24], the size of intermediate data increases with each addition of raw data in the processing. In particular, the size usually



(a) Semi-Hamilton path (b) Result for S-TSP (c) Result for CC-TSP (d) Result for DS-TSP

(e) Initial path in SP-TSP (f) Detour the path in SP-TSP (g) Result for SP-TSP (h) Result for DSP-TSP

Fig. 3 Examples of the routing schemes for chain-based computation.

increases linearly with the number of raw data involved in the processing. Hence, we assume $|s_k| = C + k * r$, where C and $r < |x|$ is a constant, k is an integer and usually equals to the number of raw data involved in s_k. It can be seen, the size of intermediate data on the *in-network processing path* increases step by step, and may become too large for the in-network processing to worth further detouring the path across more nodes. In other word, the *in-network processing path* finally may need to go along the shortest path to the sink without a little detour. However, at beginning it still needs to try to detour itself to pass more nodes to reduce their raw data transmissions. Thus, the *in-network processing path* under this case is actually a combination of TSP and the shortest path.

To find the optimal combination of TSP and the shortest path for the in-network processing, we design a SP-TSP scheme. The basic idea of SP-TSP is as follows.

- Enumerate each node in G, and take the shortest path from one node to the sink as the initial *in-network processing path* $P = \{n^1, n^2, \cdot\}$ (n^i represents the i^{th} node in P), if the path leads to the minimum total transmission cost (which is the sum of the cost on gathering the raw data x of all the nodes outside of P to P along the shortest paths and transmitting s to the sink along P). As shown in Fig. 3 (e).
- Update P with one of n^1's edge plus the shortest path from n^1's neighbor on the edge to the sink, if the updated P leads to lower total transmission cost; find the optimal updated P by enumerating each edge of n^1. As shown in Fig. 3 (f).
- Further update P by checking each edge of the second node in the current P, and then the third, until the last node (i.e., the sink). Fig. 3 (g) shows the result of SP-TSP in the above network.

We give the formal description of SP-TSP with Algorithm 1.

Algorithm 1 The proposed SP-TSP algorithm

Input: a WSN $G = (V, E)$
for each node $n_i \in G$ **do**
  Calculate the shortest path between n_i and sink, and record the path as $\mathcal{P}_i$.
  Calculate the total transmission cost in G with $\mathcal{P}_i$, and return $C_{all}(\mathcal{P}_i)$.
end for
$C_{all} = min\{C_{all}(\mathcal{P}_i)\}$
Record the node whose corresponding total cost is C_{all} as n^1, and record the corresponding path as $\mathcal{P}^1$.
$j = 1$
while $j \leq |V|$ **do**
  Construct set $S = 0$.
  for each n^j's neighbor node n_t^j that is not on $\mathcal{P}^j$ **do**
    Calculate the shortest path $\mathcal{P}_t$ between n_t^j and the sink.
    Construct path $\mathcal{P}'_t = \{n^1, n^2, \ldots, n^j, n_t^j\} \leftrightarrow \mathcal{P}_t$, where $* \leftrightarrow \star$ means to connect $*$ to $\star$.
    Calculate the total transmission cost with $\mathcal{P}'_t$, and return $C_{all}(\mathcal{P}'_t)$.
    if $C_{all}(\mathcal{P}'_t) < C_{all}$ **then**
      Add the corresponding node n_t^j into S.
      $C_{all} = C_{all}(\mathcal{P}'_t)$
    end if
  end for
  if $S == 0$ **then**
    Break, and **return:** $\mathcal{P}^j$.
  end if
  j++.
  Record the node in S whose corresponding total cost is C_{all} as n^j, and the corresponding path as $\mathcal{P}^j$.
end while

A key issue in Algorithm 1 is the calculation of the total transmission cost in G with a given *in-network processing path*. To calculate it, it is needed to ascertain

the corresponding optimal routes for other nodes outside of the path. Since the intermediate data's size increases with the processing, the optimal route for nodes outside the *in-network processing path* is not just the shortest path to it, which is different from the above two cases.

To this end, we notice that: the size of the intermediate data on the *in-network processing path*, is determined only by the number of nodes contributing their raw data to the path. Hence, we can regard that: once a node's raw data is delivered to the *in-network processing path*, it will be processed to be an independent virtual data with size r, and the virtual data will be directly sent to sink along the path without being further processed.

Thus, to calculate the total transmission cost in G with a given *in-network processing path*, we separately calculate the transmission cost for the delivery of each raw data from its source node to the sink, including the raw data transmission stage and the virtual data transmission stage. To minimize the cost of raw data transmission stage of a node (say n_i), we need to select the optimal node (say n_j) on the *in-network processing path* to process n_i's raw data, and the delivery of n_i's raw data must follow the shortest path from n_i to n_j. In this way, the optimal routing for n_i's raw data transmission stage is obtained, and the total cost for the delivery of n_i's raw data to the sink via n_j can be calculated. We present the procedure on calculating the total transmission cost in G with a given *in-network processing path* with *Algorithm* 2.

Algorithm 2 Calculation of the total transmission cost with a given *in-network processing path*

1: **Input:** the *in-network processing path* $\mathcal{P} = \{n_1, n_2, \ldots, n_k = sink\}$
2: Initialize total transmission cost in WSN $C_{all}(\mathcal{P}) = 0$
3: **for** each node $n_i \in V - \{n_1, n_2, \ldots, n_k = sink\}$ **do**
4: **for** each node $n_j \in \mathcal{P}$ **do**
5: Calculate the length L_{ij} of the shortest path between n_i and n_j in G.
6: Calculate the length l_{jk} of the path between n_j and n_k in $\mathcal{P}$.
7: Calculate the cost for delivery n_i's raw data to sink via n_j as $C^j_{n_i} = |x| \cdot L_{ij} + r \cdot l_{jk}$.
8: **end for**
9: Record $C_{n_i} = min\{C^j_{n_i}\}$, $j \in [1, k]$, and ascertain the corresponding node on $\mathcal{P}$.
10: Record the shortest path between n_i and the ascertained node on $\mathcal{P}$ as the route for n_i's raw data delivery.
11: $C_{all}(\mathcal{P}) = C_{all}(\mathcal{P}) + C_{n_i}$
12: **end for**
13: **Return:** $C_{all}(\mathcal{P}) = C_{all}(\mathcal{P}) + C \cdot l_{1k} + r \cdot \sum_{m=1}^{k-1} l_{mk}$

Since SP-TSP is based on greedy strategy which may lead to local optimum, we additionally design a DS-based SP-TSP routing scheme (named as DSP-TSP). The basic idea of DSP-TSP is as follows. Considering that dominant nodes can spread nearly evenly in the network, if path P in SP-TSP is also allowed to detour its way to pass by a nearby dominant node via the shortest path to the node, it may help SP-TSP to jump out of the local optimum, thus possibly achieving better performance. It can be seen, the detour strategy of DSP-TSP is a tradeoff between the coarse-grained searching strategy of CDS-TSP (with unit of only dominant node) and the fine-grained searching strategy of SP-TSP (with unit of neighboring node). Fig. 3 (h) shows the result of DSP-TSP in the above network.

4.3 Routing for tree-based computation partition pattern

In tree-based computation, as any two intermediate data can be in-network processed together, a tree-based routing scheme can be designed for the processing in WSNs with little constraint. Finding the optimal in-network processing tree is similar to the optimal aggregation tree problems [8] [10] [11], and have been proved to be NP-hard [20] [11]. However, considering the typical rules on the change of data sizes in the processing of many matrix computation, we can design some special routing schemes customized for the in-network processing. In this section, we show how to design such routing according to different relationships between $|s|$ and $|x|$.

Note that, for the case of $|s| = |x|$, it is well known that the optimal in-network processing tree with minimum total transmission cost is the MST. And, for the case that $|s|$ always equals to the sum of all input data's sizes, the optimal in-network processing tree is the shortest path tree (SPT). As for the other cases, we discuss the routing design as follows.

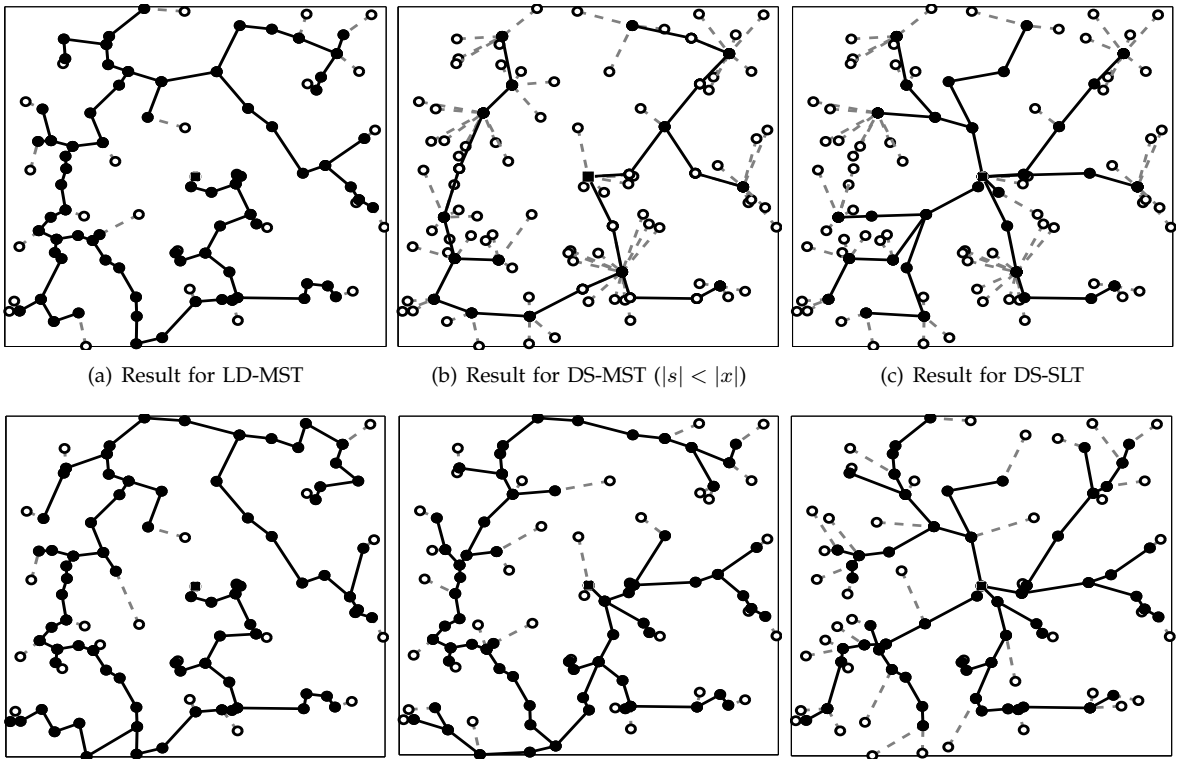
4.3.1 *Case 1: $|s|$ is constant and $|s| < |x|$*

Case analysis: For the case that $|s|$ is constant and $|s| < |x|$, the leaf nodes (transmitting x) in an in-network processing tree always have higher transmission cost than that of intermediate nodes (transmitting s). Hence, to construct an efficient in-network processing tree, a spanning tree with less leaf nodes is preferred. In other word, the optimal in-network processing tree for this case is a tradeoff between MST and a spanning tree with the minimum number of leaves.

To find an efficient tradeoff between MST and the spanning tree with minimum number of leaves, we propose a leaf deletion scheme based on MST (named as LD-MST). The basic idea of LD-MST is as follows.

- Construct the MST in G (e.g., with the classic Prim algorithm) and regard the MST as the initial in-network processing tree.
- For each leaf node n_i in the current in-network processing tree, set the neighboring leaf node n_j to be n_i's new parent if the corresponding change of total transmission cost (including the changes of n_i's cost, n_j's cost and n_i's former parent's cost) is negative and maximum.
- Re-execute the above operation until there is no cost improvement. Fig. 4 (a) shows the result of LD-MST in the above network.

For comparison, we also design a computation-constrained MST scheme (called as CC-MST) based on



(a) Result for LD-MST (b) Result for DS-MST ($|s| < |x|$) (c) Result for DS-SLT

(d) Result for CC-MST ($|s| < |x|$) (e) Result for CC-MST ($|s| \geq |x|$) (f) Result for CC-SPT ($|s|$++)

Fig. 4 Examples of the routing schemes for tree-based computation.

the Prim algorithm. Instead of always choosing the edge with the minimum weight in Prim, CC-MST always chooses the edge with the minimum "*computation-dependant weight*". We give the description of CC-MST with Algorithm 3.

Algorithm 3 The proposed CC-MST algorithm

Input: a WSN $G = (V, E)$
Denote $T = (V_T, E_T)$ as the routing tree, and initiate $V_T = \{sink\}$, $E_T = 0$.
while $V_T \neq V$ **do**
  Initiate edge set $E_l = 0$, $E_p = 0$.
  Add each edge that is in $E - E_T$ and connected to a leaf node in T, into E_l.
  Add each edge that is in $E - E_T$ and connected to a non-leaf node in T, into E_p.
  Find the optimal edge $e_l^* \in E_l$ that $|x| \cdot w_l^* - w_T(|x| - |s|)$ is the minimum, where w_l^* is weight of e_l^*, and w_T is the weight of the edge in T connected to e_l^*.
  Find the edge $e_p^* \in E_p$, which has the minimum weight w_p^*.
  if $|x| \cdot w_p^* < |x| \cdot w_l^* - w_T(|x| - |s|)$ **then**
    $E_T = E_T \cup \{e_p^*\}$;
    Add the node connected to e_p^* into V_T.
  else
    $E_T = E_T \cup \{e_l^*\}$;
    Add the node connected to e_l^* into V_T.
  end if
end while
Return: $T = (V_T, E_T)$

The reason of leveraging "*computation-dependant weight*" in CC-MST is that: when a leaf node changes to be a parent node in the in-network processing tree, the node's edge weight should also change. The result of CC-MST is shown in Fig. 4 (d).

4.3.2 Case 2: $|s|$ is constant and $|s| \geq |x|$

Case analysis: For the case that $|s|$ is constant and $|s| \geq |x|$, the intermediate nodes in an in-network processing tree always have higher transmission cost than that of leaf nodes. Hence, to construct the efficient in-network processing tree, a spanning tree with more leaf nodes is preferred. In other word, the optimal in-network processing tree for this case is a tradeoff between MST and a spanning tree with the maximum number of leaves.

To find an efficient tradeoff between MST and the spanning tree with maximum number of leaves, we propose a DS plus MST based scheme (named as DS-MST). The basic idea of DS-MST is as follows.

- Find a MDS in G, and then construct a complete graph G' consisting of all the dominant nodes and the sink, and each edge in G' represents the shortest path between the corresponding dominant nodes in G.
- Find the MST in G', the shortest paths in G corresponding to the edges in the MST of G' serve as the in-network processing paths;
- With the in-network processing paths, nodes outside of the paths in G, i.e, the dominated nodes, work as leaf nodes and send their raw data to their dominant

nodes. Fig. 4 (b) shows the result of DS-MST in the above network.

For comparison, we also employ the CC-MST algorithm designed above (though $|s| < |x|$ in that case). The result is shown in Fig. 4 (e).

4.3.3 Case 3: $|s|$ increases with the processing

Case analysis: For the case that $|s|$ increases with the processing (i.e., the size of intermediate data increases with each addition of raw data in the processing), the intermediate nodes' cost is not constant but depends on the number of their descendant nodes. We are concerned with the usual case of linear increment of $|s|$, i.e., $|s_k| = C + k * r$ (e.g., tree-based computation of SVD [1]). For this case, each intermediate node can be regarded to additionally deliver a virtual data, which has size r and cannot be further processed, along the routing to the sink. Based on this understanding, setting the shortest paths to the sink for some intermediate nodes as their routing instead of MST can possibly reduce the total transmission cost. Hence, the optimal in-network processing tree for the case of linear increment of $|s|$ is a tradeoff between MST and SPT, which is similar to the shallow light tree (SLT) [30].

To find the appropriate SLT for in-network processing with the computation constraint, we need to discuss the relationship between $|x|$ and $|s_1|$ (i.e., the value of $|s|$ at the beginning of the processing). Firstly, for the case that $|x| = |s_1|$ (e.g., in-network processing of SVD [1]), we prove, in the Appendix, that directly using SLT with certain parameter as the routing can achieve approximation ratio $1 + \sqrt{2}$. Secondly, for the case that $|x| < |s_1|$, we give the appropriate SLT design as follows. As for the case that $|x| > |s_1|$, we omit the corresponding SLT design for brevity in this paper.

Since $|x| < |s_1|$ and $|s|$ increases with the processing, $|s|$ will be always larger than $|x|$. Hence, a routing tree with more leaf nodes is preferred. Thus, we design DS-SLT scheme based on the DS-MST proposed above. The basic idea of DS-SLT is as follows. Firstly, we employ DS-MST proposed above to construct a spanning tree in G. Then, the algorithm for establishing SLT is performed on the spanning tree, i.e., conducting deep first searching (DFS) within the spanning tree: if an intermediate node is becoming not "*shallow*" to the sink, the path from the node to the sink in the spanning tree will be replaced by the path from the node to the sink in SPT. Metric for "*shallow*" here is defined as ratio α of the distance from the node to sink in the spanning tree to that in SPT. Fig. 4 (c) shows the result of DS-SLT in the above network. The value of parameter α depends on the ratio of the MST's cost to SPT's cost as well as the ratio of the initial size of s to r.

For comparison, we additionally design a special computation-constrained SPT scheme (called as CC-SPT), as shown in Algorithm 4. The reason of leveraging "weight of path" in Algorithm 4 is that the traffic on a path will increase with r for each addition of a node during the algorithm. The results of CC-SPT are shown in Fig. 4 (f).

Algorithm 4 The proposed CC-SPT algorithm

Input: a WSN $G = (V, E)$
Denote $T = (V_T, E_T)$ as the routing tree, and initiate $V_T = \{sink\}$, $E_T = 0$.
while $V_T \neq V$ **do**
  Initiate edge set $E_l = 0$, $E_p = 0$.
  Add each edge that is in $E - E_T$ and connected to a leaf node in T, into E_l.
  Add each edge that is in $E - E_T$ and connected to a non-leaf node in T, into E_p.
  Find the optimal edge $e_l^* \in E_l$ that $|x| \cdot w_l^* - w_T(|x| - |s|) + d_l^* \cdot r$ is the minimum, where d_l^* is the weight of path between the sink and the node in T connected to e_l^*.
  Find the edge $e_p^* \in E_p$ that $|x| \cdot w_p^* + d_p^* \cdot r$ is the minimum.
  if $|x| \cdot w_p^* + d_p^* \cdot r < |x| \cdot w_l^* - w_T(|x| - |s|) + d_l^* \cdot r$ **then**
    $E_T = E_T \cup \{e_p^*\}$;
    Add the node connected to e_p^* into V_T.
  else
    $E_T = E_T \cup \{e_l^*\}$;
    Add the node connected to e_l^* into V_T.
  end if
end while
Return: $T = (V_T, E_T)$

4.3.4 Case 4: $F_3(\{s\})$ only at sink

Sometimes, function $F_3(\{s\})$ may take high computation cost (e.g., diff-RLS in [29]), which makes $F_3(\{s\})$ not available to be executed on resource-limited sensor nodes except for the sink node. Under this circumstance, the centralized algorithm has to be partitioned into multiple chain-based computations, and the final results of these chain-based computations will be "merged" with function $F_3(\{s\})$ at the sink node. Optimal routing design for this case can be formulated into the classic multiple TSP problem (MTSP) (if $|s| = c$). As for the case $|s| + +$, referring to the above SP-TSP, a multiple SP-TSP algorithm (could be called as MSP-TSP) can be designed. The details are omitted for briefness.

5 Simulations

In this section, we conduct extensive simulations of the proposed schemes. In the simulations, we use random-generated networks by randomly deploying N nodes in area Z with size $L * L$. The transmission range of each node is denoted by R. The weight of each edge between neighboring nodes is regarded to be proportional to the edge's length.

To extensively evaluate the proposed schemes' performance in different network densities, network scales and relative data sizes (e.g., $|s|/|x|$, $r/|x|$ or $|x|/C$), we conduct the simulations in two scenarios. First, we fix the number of nodes to be 100 but gradually increase the network density by increasing R from 15 to 45. The area

in this scenario is fixed to be $120 * 120$. For each network density, we gradually increase the corresponding relative size from 0.2 to 1.

Then we maintain the network density with fixing $R = 30$ but gradually increase the network scale by increasing the number of nodes from 100 to 500. Note that, to maintain the network density, when increasing the number of nodes, the size of the deployment area also needs to be increased. Also, for each network scale, the relative size increases from 0.2 to 1. In both scenarios, 20 simulations are performed for each parameter set, and the average value of each scheme's transmission cost is calculated.

5.1 Baseline: Typical Straightforward Schemes

To evaluate the performance of the proposed schemes, a baseline needs to be set. Although there are lots of routing algorithms proposed for WSNs with in-network processing, as analyzed in Section 2, the routing algorithms are designed with assumption of common processing function and ignoring the difference in the input data types. In lossless in-network processing of a designated centralized algorithm, either the processing functions are not unique, or the input data is of different types, making the existing routing algorithms not applicable here.

To set the baseline, we use straightforward routing idea for comparison. Meanwhile, for chain-based in-network processing, we directly regard the longest shortest path as the *in-network processing path*, and the routes for nodes outside of the path are just the shortest paths between the nodes to the *in-network processing path*. As for tree-based in-network processing, we directly use SPT as the baseline.

Fig. 5 compares the performance between the baseline schemes and the proposed routing schemes. It can be seen, the proposed schemes generally far outperform the baseline schemes, for various cases of lossless in-network processing. In the following, we further give an extensive comparison between the proposed routing schemes at different network and computation conditions. We find that, the proposed schemes perform well at certain conditions. This helps us to appropriately select the routing scheme according to a given network and computation condition of the lossless in-network processing in practice.

5.2 Simulations for chain-based in-network processing

Fig. 6 shows the simulation results of S-TSP and CC-TSP at different network and computation conditions. According to Fig. 6 (a)(b), we find that S-TSP always has lower total transmission cost than CC-TSP. And, the advantage of S-TSP becomes more prominent in sparse network (with lower transmission range) or large-scale network.

Fig. 7 shows the simulation results of DS-TSP and CC-TSP. According to Fig. 7 (a)(b), CC-TSP outperforms DS-TSP in most cases, especially when x has similar size to s or the network scale is large. However, we notice that, smaller network scale will mitigate the performance gap between DS-TSP and CC-TSP. Moreover, when $|x|$ is much smaller than $|s|$ and the network density is high, DS-TSP could lead to lower total transmission cost than CC-TSP.

Fig. 8 shows the simulation results of SP-TSP and DSP-TSP. According to Fig. 8 (a), SP-TSP and DSP-TSP perform well at different conditions, respectively. For dense WSN and small increment of $|s|$, SP-TSP tends to outperform DSP-TSP. However, when the density decreases or the increment of $|s|$ becomes large, DSP-TSP has better performance than SP-TSP. According to Fig. 8 (b), the comparison betwenn SP-TSP and DSP-TSP becomes much sensitive to the increment of $|s|$. When the increment of $|s|$ is small, SP-TSP far outperforms DSP-TSP in large scale network. However, when the increment of $|s|$ is large, DSP-TSP can far outperform SP-TSP in large scale network.

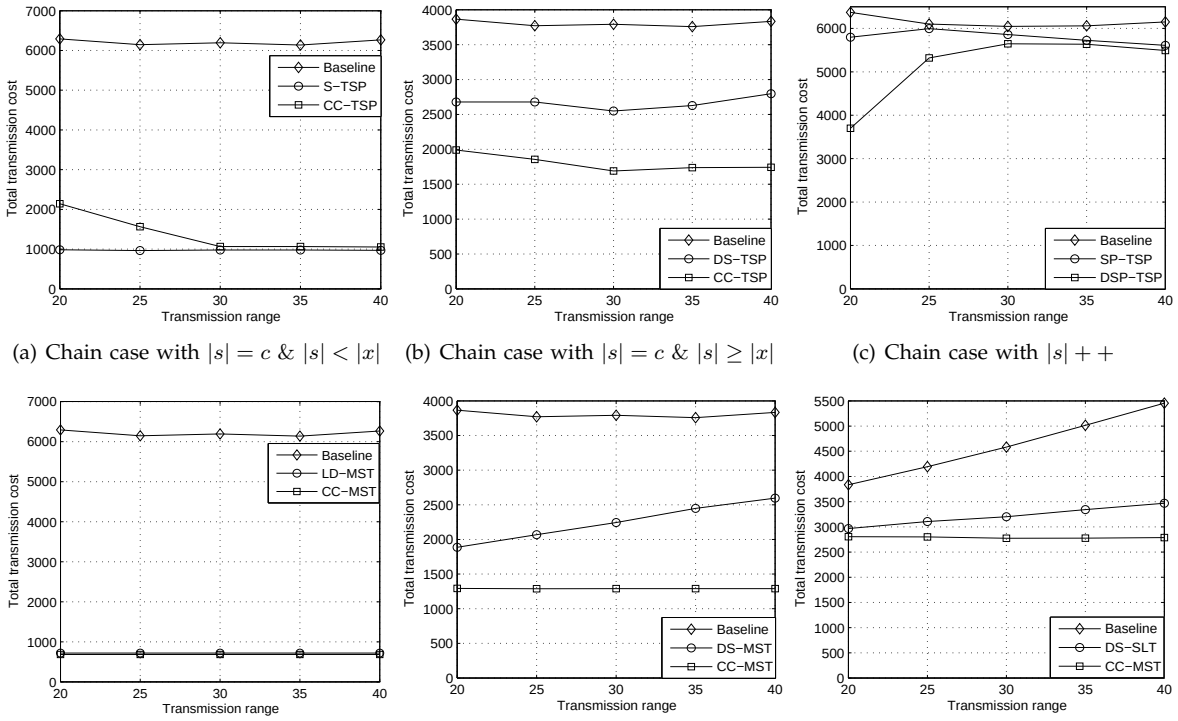
5.3 Simulations for tree-based in-network processing

Fig. 9 shows the comparison between LD-MST and CC-MST. It can be seen, the performance of the two schemes is almost the same at different conditions. Generally, CC-MST performs a little better than LD-MST. In addition, we notice that the performance of the two schemes is independent of the network density.

Fig. 10 shows the comparison between DS-MST and CC-MST. It can be seen, DS-MST and CC-MST perform well at different conditions, respectively. For small $|x|$, DS-MST tends to outperform CC-MST. However, when $|x|$ becomes close to $|s|$, CC-MST has better performance than DS-MST. The comparison result is independent of the network density. According to Fig. 10 (b), DS-MST outperforms CC-MST when $|x|$ is much smaller than $|s|$, no matter how large the network scale is. However, when $|x|$ is close to $|s|$, CC-MST will has better performance than DS-MST. That is say, the comparison result is also independent of the network scale.

Fig. 11 shows the comparison between DS-SLT and CC-SPT. It can be seen, the comparison between DS-SLT and CC-SPT is a little similar to that between DS-MST and CC-MST. For small initial size of s, DS-SLT tends to outperform CC-SPT, no matter how large the network density or network scale is. However, when the initial size of s becomes close to $|x|$, CC-SPT has better performance than DS-SLT.

To sum up, the performance of the proposed computation-constrained routing schemes much depends on the network scale, density and relative sizes for x and s. Different routing ideas perform well only at certain conditions. We need to appropriately choose the



(a) Chain case with $|s| = c$ & $|s| < |x|$ (b) Chain case with $|s| = c$ & $|s| \geq |x|$ (c) Chain case with $|s| + +$

(d) Tree case with $|s| = c$ & $|s| < |x|$ (e) Tree case with $|s| = c$ & $|s| \geq |x|$ (f) Tree case with $|s| + +$

Fig. 5 Comparison between the baselines and the proposed schemes.

routing scheme according to the characteristics of both the network and the computations.

In addition, it is worth mentioning that the computation cost of the routing schemes is also an important issue. However, compared to the high-power data transmission, the low-power computation is generally a minor role in energy consumption in most cases, which inspires many works (e.g., data aggregation) to employ computation to reduce the data transmission. Since lower total transmission cost generally implies lower total energy consumption, the comparison of transmission cost among the routing schemes in this section, actually also represents the comparison of energy consumption among them.

Finally, we give a comparison between chain-based routing and tree-based routing. As multiple intermediate data can be processed together in tree-based routing, the routes do not need to detour their way too much as the chain-based path does. Hence, the end-to-end delay of tree-based routes is surely lower than that of chain-based routes. As for the transmission cost, with less constraint on selecting destination nodes in the routing, tree-based routes generally have lower transmission cost than chain-based routes, especially for the case of the increasing intermediate data's size. Only when the intermediate data' size is always smaller than that of raw data, the chain-based routes may be possible to have lower transmission cost than tree-based routes, as the former may have less leaf nodes (i.e., less raw data transmission). In addition, it should be noted that tree-based partition pattern is not always feasible for a centralized algorithm (e.g., LSE in [17]).

6 Conclusions

Lossless in-network processing is required in many domain specific monitoring applications of WSNs. This paper gives a framework on lossless in-network processing of a given complicated centralized algorithm in resource-limited WSNs, with introducing some general patterns on the computation partitioning. Constrained by the patterns, a series of special routing algorithms for different cases of the computation results' sizes and network parameters, are introduced and compared. These routing algorithms can serve as guidelines not only for efficient lossless in-network processing of the practical engineering algorithms in applications of large object monitoring and diagnosis with WSNs, but also for distributed computing of big data in Internet.

Acknowledgement

The work presented in this paper was supported in part by the NSF of China with Grant 61572217 and 61572218. Xuefeng Liu is the corresponding author.

Appendix

For the case that $|s|$ linearly increases with the in-network processing (i.e., $|s_k| = C + k * r$) and initially $|s| = |x|$ (e.g., distributed computation of SVD [1]), we formulate the problem of finding the optimal in-network processing tree (denoted as *OIT problem*) as follows.

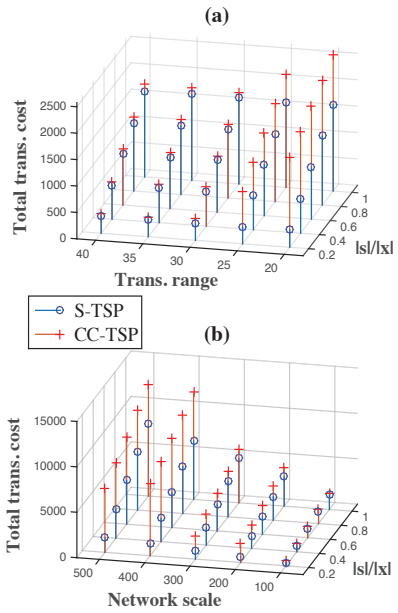


Fig. 6 Performance of S-TSP and CC-TSP w.r.t. different (a) density and $|s|/|x|$; (b) scale and $|s|/|x|$.

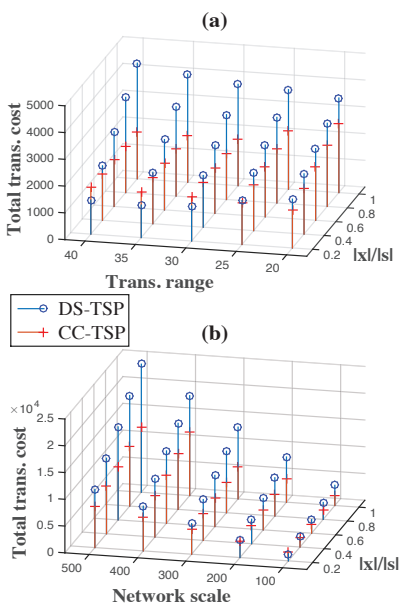


Fig. 7 Performance of DS-TSP and CC-TSP w.r.t. different (a) density and $|x|/|s|$; (b) scale and $|x|/|s|$.

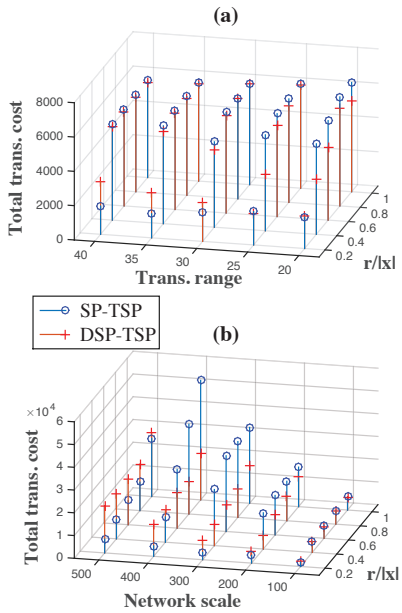


Fig. 8 Performance of SP-TSP and DSP-TSP w.r.t. different (a) density and $r/|x|$; (b) scale and $r/|x|$.

- ***Given***: *a network $G = (V, E)$, where V is the set of nodes and E is the set of edges between neighboring nodes. Initially, each node has raw data with equal size $C + r$.*
- ***Assume***: *either raw data or intermediate data can be in-network processed by any node, and the processing result's size is $C + (m + 1)r$, where m is the number of descendant nodes sending data to the node.*
- ***Objective***: *construct an optimal in-network processing tree rooted by a given sink node, so as to minimize the total transmission cost in G.*

It can be seen, the OIT problem is a special case of the traditional aggregation problems which have been proved to be NP-hard [26] [27]. In the following, we present a sub-optimal tree for OIT, with which the transmission cost is no more than $1 + \sqrt{2}$ times that of the optimal tree.

Denote $T_{opt} = (V, E')$ as the optimal in-network processing tree for *OIT problem*, where E' is the set of edges in T_{opt}. Let $w(e_i)$ as the weight of edge e_i, and $C + r_i$ is the size of data transmitted on e_i, where $r_i = k_i * r$ (k_i is an integer). The total transmission cost of the in-network processing on T_{opt} can be denoted as:

$$C_{opt} = \sum_{e_i \in E'} (C + r_i) w(e_i) \tag{1}$$

We first give a lower bound on C_{opt} as follows.

LEMMA 1. *The cost of using an optimal in-network processing tree for OIT problem is bounded from below by $C_{opt} \geq C \cdot c_{MST} + r \cdot c_{SSP}$, where c_{MST} is the cost of the minimum spanning tree (MST) of all nodes in V (i.e., the sum of all edges' weights in MST), and c_{SSP} is the sum of the costs of all the shortest paths to the sink node.*

Proof: According to Equation 1, we have:

$$C_{opt} = \sum_{e_i \in E'} (C + r_i) w(e_i) = C \cdot \sum_{e_i \in E'} w(e_i) + \sum_{e_i \in E'} r_i \cdot w(e_i) \tag{2}$$

It can be seen, there are two parts of cost in C_{opt}. One is $C \cdot \sum_{e_i \in E'} w(e_i)$, and another is $\sum_{e_i \in E'} r_i \cdot w(e_i)$. Meanwhile, $\sum_{e_i \in E'} w(e_i)$ is the cost of T_{opt}. Since T_{opt} contains all nodes in V, its cost must be no less than c_{MST}, i.e., $\sum_{e_i \in E'} w(e_i) \geq c_{MST}$.

In addition, according to the processing model, the variant part of the data size, i.e., r_i, is cumulated directly during the in-network processing. Therefore, $\sum_{e_i \in E'} r_i \cdot w(e_i)$ is equivalent to the cost for that all sensor nodes transmit their data with size r to the sink along T_{opt} without any processing. It is well known that, gathering all nodes' data to the sink without processing along a shortest path tree will have the minimum cost. Hence, $\sum_{e_i \in E'} r_i \cdot w(e_i)$ must be no less than $r \cdot c_{SSP}$. Therefore, the lemma follows immediately from what we have proved. □

Since finding T_{opt} is NP-hard, we propose a sub-optimal tree for OIT problem. In particular, we establish a *shallow light tree* (SLT) [23] [30] with certain parameter as the sub-optimal tree. SLT is a spanning tree that balances the performance of the SPT ("shallow") and the MST ("light"). The basic idea of constructing SLT is simple: conducting deep first searching (DFS) within MST, if a node is becoming not *"shallow"* to the root, the path from the node to the sink in MST will be replaced by the path from the node to the sink in SPT. Metric for *"shallow"* is defined as the ratio α of the distance from the node to root in MST to that in SPT.

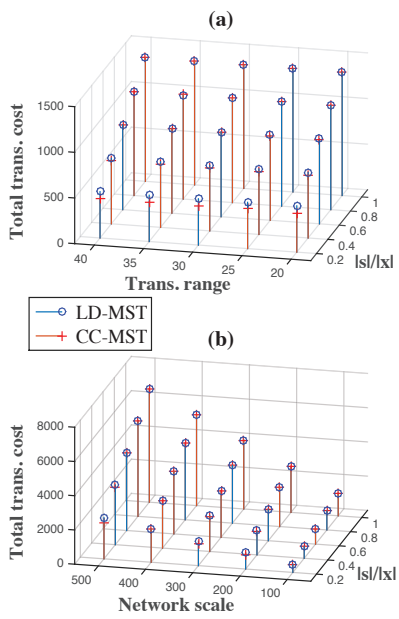


Fig. 9 Performance of LD-MST and CC-MST w.r.t. different (a) density and $|s|/|x|$; (b) scale and $|s|/|x|$.

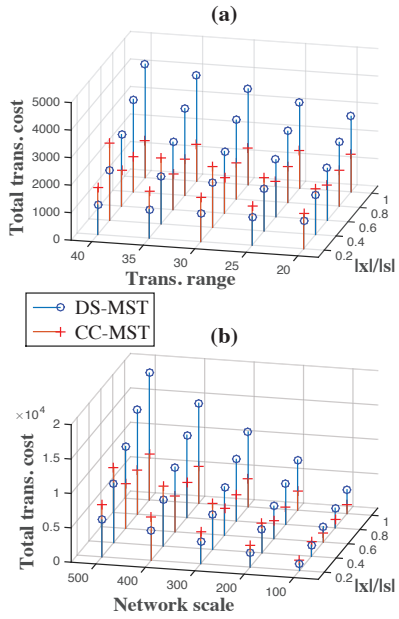


Fig. 10 Performance of DS-MST and CC-MST w.r.t. different (a) density and $|x|/|s|$; (b) scale and $|x|/|s|$.

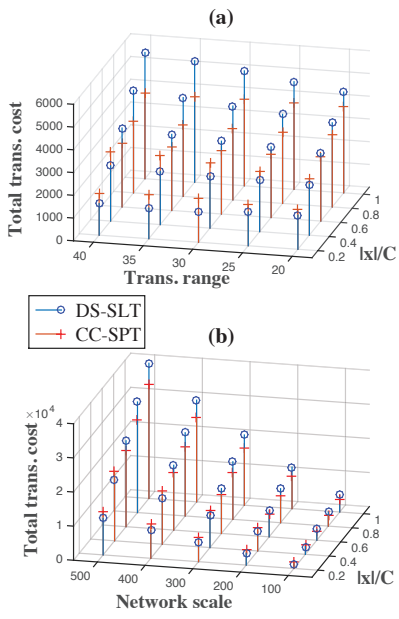


Fig. 11 Performance of DS-SLT and CC-SPT w.r.t. different (a) density and $|x|/C$; (b) scale and $|x|/C$.

To find the optimal SLT, we need to calculate the optimal parameter α of SLT in a WSN with arbitrary topology. According to [30], given a graph $G = (V, E)$ and a number $\alpha > 1$, a SLT has the following two properties:

- The distance between any node and the root in the SLT is no more than α times the length of the shortest path from that node to the root in G.
- The total cost of a SLT c_{SLT} is no more than $\frac{\alpha+1}{\alpha-1}$ times that of the MST of the graph G;

Theorem 1. *The cost of using a SLT with parameter $\alpha = 1+\sqrt{2}$ (denoted by $SLT_{1+\sqrt{2}}$) for* OIT problem *is no more than $1+\sqrt{2}$ times that of the optimal in-network processing tree.*

Proof: Since the variant part r_i of the data size is cumulated during the computation in $SLT_{1+\sqrt{2}}$ while the constant part C of the data size is maintained on each edge in $SLT_{1+\sqrt{2}}$, the total cost of using $SLT_{1+\sqrt{2}}$ for *OIT problem* can be expressed as

$$\begin{aligned} C_{SLT(1+\sqrt{2})} &= C \cdot c_{SLT_{1+\sqrt{2}}} + \sum_{u_i \in V} r \cdot path(u_i, t)|_{SLT_{1+\sqrt{2}}} \\ &\leq C \cdot \frac{1+\sqrt{2}+1}{1+\sqrt{2}-1} c_{MST} + r \cdot (1+\sqrt{2}) c_{SSP} \\ &= C \cdot (1+\sqrt{2}) c_{MST} + r \cdot (1+\sqrt{2}) c_{SSP} \\ &\leq (1+\sqrt{2}) C_{opt} \end{aligned} \quad (3)$$

Therefore, the theorem follows immediately from what we have proved.

□

Note that, $\alpha = 1+\sqrt{2}$ is not the optimal parameter for SLT to minimize the transmission cost, considering the arbitrary topology of the WSN. We expect to calculate the optimal α based on the characteristic of the WSN's topology. According to Equation 3,

$$C_{SLT_\alpha} \leq C \cdot \frac{\alpha+1}{\alpha-1} c_{MST} + r \cdot \alpha c_{SSP} \quad (4)$$

Hence, the optimal parameter $\alpha^* = 1 + \sqrt{2\frac{C}{r} \cdot \frac{c_{MST}}{c_{SSP}}}$, and $C_{SLT_{\alpha^*}} \leq C_{SLT_{1+\sqrt{2}}} \leq (1+\sqrt{2}) C_{opt}$.

REFERENCES

[1] Xuefeng Liu, Jiannong Cao, Wen-Zhan Song, Peng Guo, Zongjian He, *Distributed Sensing for High-Quality Structural Health Monitoring using WSNs*, IEEE Transactions on Parallel and Distributed Systems, Vol. 26, No. 3, pp. 738-747, 2015.

[2] G. Kamath, L. Shi, and W.-Z. Song, *Component-average based distributed seismic tomography in sensor networks*, in DCOSS. IEEE, 2013, pp. 88-95.

[3] H. T. Pham and B.-S. Yang, *Estimation and forecasting of machine health condition using arma/garch model*, Mechanical Systems and Signal Processing, vol. 24, no. 2, pp. 546-558, 2010.

[4] Y.-F. Huang, S. Werner, and et al., *State estimation in electric power grids: Meeting new challenges presented by the requirements of the future grid*, Signal Processing Magazine, IEEE, vol. 29, no. 5, pp. 33-43, 2012.

[5] Xuefeng Liu, Jiannong Cao, Wen-Zhan Song, Shaojie Tang, *Distributed Sensing for High Quality Structural Health Monitoring Using Wireless Sensor Networks*, in IEEE RTSS, Dec. 2012, pp. 75-84.

[6] B. Yu, J. Li, and et al., *Distributed data aggregation scheduling in wireless sensor networks*, in INFOCOM, 2009, pp. 2159-2167.

[7] Fasolo, E. ; DEI, Padova Univ. ; Rossi, M. ; Widmer, J. ; Zorzi, M., *In-network Aggregation Techniques for Wireless Sensor Networks: A Survey*, IEEE Wireless Communications, vol. 14, no. 2, 2007, pp. 70-87.

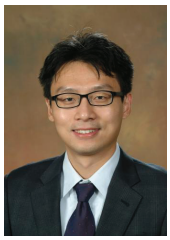
[8] Cunqing Hua and Tak-Shing Peter Yum, *Optimal Routing and Data Aggregation for Maximizing Lifetime of Wireless Sensor Networks*, IEEE/ACM Transactions on Networking, vol. 16, no. 4, 2008, pp. 892-903.

[9] Sharanya Eswaran, Matthew Johnson, Archan Misra, Thomas La Porta, *Adaptive In-Network Processing for Bandwidth and Energy Constrained Mission-Oriented Multihop Wireless Networks*, IEEE Transactions on Mobile Computings, Vol. 11, No. 09, 2012, pp: 1484-1498.
[10] Yan Wu ; Dept. of Comput. Sci., Purdue Univ., West Lafayette, IN ; Fahmy, S. ; Shroff, N.B., *On the Construction of a Maximum-Lifetime Data Gathering Tree in Sensor Networks: NP-Completeness and Approximation Algorithm*, in INFOCOM, Phoenix, AZ, Apr. 2008.
[11] Weifa Liang, and Yuzhen Liu, *Online Data Gathering for Maximizing Network Lifetime in Sensor Networks*, IEEE Transactions on Mobile Computing, vol. 6, no. 1, 2007, pp. 2-11.
[12] XiaoHua Xu, Mo Li, XuFei Mao, Shaojie Tang, *A Delay-Efficient Algorithm for Data Aggregation in Multihop Wireless Sensor Networks*, IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 1, 2011, pp. 163-175.
[13] Peng-Jun Wan, Scott C.-H., Lixin Wang, Zhiyuan Wan, Xiaohua Jia, *Minimum-latency aggregation scheduling in multihop wireless networks*, In MobiHoc, New York, USA, 2009, pp. 185-194.
[14] Bo Hong and Viktor K. Prasanna, *Optimizing a Class of In-network Processing Applications in Networked Sensor Systems*, The 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2004), October 2004.
[15] J. Juang and R. Pappa, *Eigensystem realization algorithm for modal parameter identification and model reduction*, Journal of Guidance, Control, and Dynamics, vol. 8, no. 5, pp. 620-627, 1985.
[16] Schizas, I.D. ; Dept. of Electr. and Comput. Eng., Univ. of Minnesota, Minneapolis, MN ; Mateos, G. ; Giannakis, G.B., *Distributed LMS for Consensus-Based In-Network Adaptive Processing*, IEEE Transactions on Signal Processing, Vol. 57, No. 6, 2009, pp. 2365-2382.
[17] Ali H. Sayed, Cassio G. Lopes, *Distributed Recursive Least-squares Strategies Over Adaptive Networks*, Fortieth Asilomar Conference on Signals, Systems and Computers (ACSSC), Oct. 2006.
[18] Liang Zhao, Wen-Zhan Song, Lei Shi, Xiaojing Ye, *Decentralized Seismic Tomography Computing In Cyber-Physical Sensor Systems*, Cyber-Physical Systems, Taylor & Francis, Vol. 1, No. 2-4, 2015, pp. 91-112.
[19] G. R. Andrews, *Foundations of parallel and distributed programming*, Addison-Wesley Longman Publishing Co., Inc., 1999.
[20] R. Cristescu, B. Beferull-Lonzano, and M. Vetterli, *On Network Correlated Data Gathering*, In Proceeding of the 23rd Conference of the IEEE Comunications Society (INFOCOM), 2004.
[21] A Jindal, M. Liu, *Networked computing in wireless sensor networks for structural health monitoring*, IEEE/ACM Transactions on Networking, vol. 20, no. 4, pp. 1203-1216, 2012.
[22] H. Zha and H. Simon, *On updating problems in latent semantic indexing*, SIAM Journal on Scientific Computing, vol. 21, p. 782, 1999.
[23] B. Awerbuch, A. Baratz, and D. Peleg. *Cost-Sensitive Analysis of Communication Protocols*. In Proc. of the 9th Symposium on Principles of Distributed Computing (PODC), 1990.
[24] H. Zha and H. Simon, *On updating problems in latent semantic indexing*, SIAM Journal on Scientific Computing, vol. 21, p. 782, 1999.
[25] S. Khuller, B. Raghavachari, N. Young, *Balancing minimum spanning trees and shortest-path trees*, Algorithmica, Vol. 14, no. 4, pp 305-321, 1995.
[26] A. Goel and D. Estrin, *Simultaneous Optimization for Concave Costs: Single Sink Aggregation or Single Source Buy-at-Bulk*, Proc. ACM/SIAM Symp. Discrete Algorithms, pp. 499-505, 2003.
[27] S. Guha, A. Meyerson, and K. Munagala, *A constant factor approximation for the single sink edge installation problem*, Proceedings of 33rd ACM Symposium on Theory of Computing, 2001.
[28] Salman, F.S., Cheriyan, J., Ravi, R., Subramanian, S., *Approximating the Single-Sink Link Installation Problem in Network Design*, SIAM Journal on Optimization, Vol. 11, No. 3, pp. 595-610, 2000
[29] F. S. Cattivelli, C. G. Lopes, and A. H. Sayed, *Diffusion recursive least-squares for distributed estimation over adaptive networks*, IEEE Transactions on Signal Processing, Vol. 56, No. 5, pp. 1865-1877, 2008.
[30] S. Khuller, B. Raghavachari, N. Young, *Balancing minimum spanning trees and shortest-path trees*, Algorithmica, Vol. 14, no. 4, pp 305-321, 1995.

Peng Guo received his M.S. and Ph.D. degree from Huazhong University of Science and Technology, Wuhan, China, in 2003 and 2008, respectively. He is currently an Associate Professor at the school of Electronic Information and Communications in Huazhong University of Science and Technology. His research interests include wireless sensor networks, distributed computing and in-network processing. He has served as a reviewer for several international journals/conference proceedings.

Xuefeng Liu received his M.S. and Ph.D. degree from Beijing Institute of Technology, China, and University of Bristol, UK, in 2003 and 2008, respectively. He is currently an Associate Professor at the school of Electronic Information and Communications in Huazhong University of Science and Technology. His research interests include wireless sensor networks and in-network processing. He has served as a reviewer for several international journals/conference proceedings.

Jiannong Cao received the MSc and PhD degrees in computer science from Washington State University, Pullman, Washington, in 1986 and 1990, respectively. He is currently the head and chair professor in the Department of Computing at Hong Kong Polytechnic University, Hong Kong. His research interests include parallel and distributed computing, mobile computing and big data analytics. He is a IEEE Fellow and a senior member of the China Computer Federation. He has served as a member of editorial boards of several international journals, a reviewer for international journals/conference proceedings, and also as an organizing/program committee member for many international conferences.



Shaojie Tang is currently an assistant professor of Naveen Jindal School of Management at University of Texas at Dallas. He received his PhD in computer science from Illinois Institute of Technology in 2012. His research interest includes social networks, e-business and optimization. Tang served as chairs and TPC members at numerous conferences.