Local Thresholding in General Network Graphs

Ran Wolff
Information Systems Department
University of Haifa
rwolf@is.haifa.ac.il

Local thresholding algorithms were first presented more than a decade ago and have since been applied to a variety of data mining tasks in peer-to-peer systems, wireless sensor networks, and in grid systems. One critical assumption made by those algorithms has always been cycle-free routing. The existence of even one cycle may lead all peers to the wrong outcome. Outside the lab, unfortunately, cycle freedom is not easy to achieve.

This work is the first to lift the requirement of cycle freedom by presenting a local thresholding algorithm suitable for general network graphs. The algorithm relies on a new repositioning of the problem in weighted vector arithmetics, on a new stopping rule, whose proof does not require that the network be cycle free, and on new methods for balance correction when the stopping rule fails.

The new stopping and update rules permit calculation of the very same functions that were calculable using previous algorithms, which do assume cycle freedom. The algorithm is implemented on a standard peer-to-peer simulator and is validated for networks of up to 80,000 peers, organized in three different topologies, which are representative of the topology of major current distributed systems: the Internet, structured peer-to-peer systems, and wireless sensor networks.

I. INTRODUCTION

Recent years have seen a surge in the number, the pervasiveness, and the capabilities of networked devices, followed by ever greater interest in efficient algorithms for distributed computation of various functions. Some of this interest is driven by the performance superiority of in-network computation. Performance becomes increasingly important as distributed data becomes abundant (e.g., when processed by apps running on smartphones), as the balance between computation and communication costs tilts in favor of the former (e.g., in a peer-to-peer environment), and when energy conservation becomes a major concern (as in a wireless sensor network). Additional causes for this interest are more social than technological: A distributed architecture – harder to manipulate and control by any single entity – is preferred where users distrust any third party, and where privacy is a concern. This is frequently the case where there is suspicion of bias (e.g., in shopping recommendations) or in semi-legitimate applications (e.g., file sharing).

In-network computation algorithms fall into several categories. Those categories which provide proof of correctness include broadcast and convergecast based algorithms such as those implemented using MapReduce [27] or tailor-made methods [19], gossip algorithms [16], [5], [14], [7], [2], [15], [8], [22], [12], and local thresholding algorithms [31], [21], [50], [4], [3]. Broadcast and convergecast require global coordination, which can be costly. When dealing with dynamically changing data and topologies, as in peer-to-peer and wireless sensor networks, such algorithms are not attractive. Gossip algorithms require no coordination and are suitable for dynamic data and topologies. However, the correctness of gossip algorithms relies on rapid mixing of the inputs, which is, by definition, communication intensive.

Local thresholding algorithms are a third category of in-network computation algorithms. Unlike gossip algorithms, they focus on decision rather than approximation problems. For instance, a basic local thresholding algorithm [31] would compute a majority vote where its counterpart gossip based algorithm computes the average. Research has shown that many data mining algorithms (e.g., associations, decision trees, approximated clustering) can be mapped to large numbers of convex thresholding decisions.

At the heart of any local thresholding algorithm lies a local stopping rule: A condition computed by each peer on its data and the messages it received and has sent. When the condition is violated, the peer must send out messages in prescribed ways. However, when the condition holds, the peer does not have to send any further messages because either every peer currently computes the correct outcome, or there is a peer in the network whose condition is violated and who is responsible for correcting the computation outcome of all peers. Since local thresholding algorithms rely on achieving local balance rather than on mixing the inputs they are the most communication thrifty of the three categories. However, all local thresholding algorithms presented until today require cycle-free routing, which makes them very difficult to implement in real distributed systems.

This work is the first to present a local thresholding algorithm suitable for general network graphs. The algorithm relies on a new local stopping rule and on new update rules that restore local balance when the stopping rule fails. Unlike those used in previous local thresholding algorithms, the new rules do not rely on cycle freedom for their correctness. The new stopping and update rules are general and can replace any of those used in existing local data mining algorithms. Additionally, because they can handle cycles, algorithms based on the new rules can handle partial message failure. Of no less importance is the representation of the problem in a new mathematical framework, which simplifies proofs and invites further development of local thresholding algorithms.

The rest of this paper is organized as follows: First, Sec-
Section II describes the notation used and the formal problem definition, as well as the success metrics. Section III provides the main mathematical contribution – the new stopping rule. Section IV complements Section III by describing a general and accurate balance correction method. Section V combines those two contributions to form a useful algorithm. Thorough experimentation is described in Section VI. Next, Section VII explains the relation of previous work to this one. We conclude with some open research questions in Section VIII.

II. NOTATIONS AND PROBLEM STATEMENT

A. Mathematical notation

Much of the math in this paper uses weighted averages. To simplify notation throughout the paper we adopt the notation proposed in [12] in which \( \oplus \) denotes the weighted average of a pair and \( \bigoplus \) the weighted average of a set of vectors. The scalar multiplication of a weighted vector, which affects its weight, is denoted \( \odot \). Formal definition follows.

**Definition 1.** [Weighted Vector Space] Let \( V \) be a vector space and \( C \) a corresponding field of scalars. Denote \( + \) and \( \cdot \) the addition and the scalar multiplication of \( V \). The weighted vector space \( W \) with addition \( \oplus \) and scalar multiplication \( \odot \) is defined as follows:

- The elements of \( W \) are pairs \( \langle \vec{v}, c \rangle \) such that \( \vec{v} \in V \) and \( c \in C \)
- The scalar field of \( W \) is \( C \)
- The scalar multiplication is defined

\[
  \langle v_1, c_1 \rangle \odot \langle v_2, c_2 \rangle = \langle v_1 \cdot c_1, v_2 \cdot c_2 \rangle
\]

- The addition is defined

\[
  \langle v_1, c_1 \rangle \oplus \langle v_2, c_2 \rangle = \langle v_1 + \frac{c_1}{c_1 + c_2} \cdot v_1, v_2 + \frac{c_2}{c_1 + c_2} \cdot v_2 \rangle
\]

To be precise, \( \frac{c_1}{c_1 + c_2} \) denotes the multiplication of \( c_1 \) in the scalar inverse of \( c_1 + c_2 \). The obvious example for a weighted vector space is one in which \( V = \mathbb{R}^d \) and \( C = \mathbb{R} \). However, the definition of the weighted vector space is general enough to include many types of \( V \) and \( C \). Of special interest is that for a specific space of random vectors \( V \) the corresponding \( C \) can be the space of covariance matrices. This has many applications in data mining and machine learning, including z-score normalization.

It is easy to validate that the weighted vector space \( W \) with the operations \( \odot \) and \( \oplus \) is a vector space and that any \( X_0 \) whose weight is the zero element of \( C \) is an identity element of this space. Also, the triangle inequality with respect to the \( L2 \) norm \( ||\cdot|| \) holds for the vector part of the weighted vector.

For brevity, we make four additional notations: We will refer to the vector part of any \( X \in W \) as \( \vec{X} \) and to its scalar part as \( |X| \). Additionally, the additive inverse operator is denoted by \( \ominus \) where \( X \ominus Y = Z \) if and only if \( X = Y \ominus Z \). Finally, for a set \( S = \{X_1, X_2, \ldots, X_n : X_i \in W\} \) the additive iteration \( X_1 \oplus X_2 \oplus \cdots \oplus X_n \) is shorthanded to \( \bigoplus_{X_i \in S} X_i \).

B. Problem statement

Let \( P = \{p_1, \ldots, p_n\} \) be a set of peers with inputs \( X = \{X_1, \ldots, X_n\} \), respectively. Let \( N_i \subseteq P \) be the set of peers connected to \( p_i \). The average input is \( \bigoplus_{p_i \in P} X_i \) or just \( \bigoplus X \) for short. The objective of the peers is to all compute a function \( f(\bigoplus X) \). Peers communicate by sending messages, where each message consists of a single weighted vector. The latest message sent by \( p_i \) to a neighbor \( p_j \in N_i \) is denoted \( X_{i,j} \).

Throughout this paper it is assumed that both the set of peers, \( P \), their inputs, \( X \), and the connectivity of each peer, \( N_i \), vary over time. Link failures are modeled as changes in the neighbor sets of the peers at both ends of the link and peer failure as failure of all its links. The method of failure detection is not specified and it is sufficient that failures are eventually detected (i.e., a heartbeat mechanism is sufficient.) This paper assumes symmetric communication, i.e., that \( p_j \in N_i \iff p_i \in N_j \). Our proofs of correctness also assume communication is ordered and reliable. However, we show (in Section V) how sufficient ordering can be enforced. We further show experimentally that limited random message dropping does not affect correctness in any serious way.

Let \( f_i \) be a function which determines the output of peer \( p_i \). An algorithm provides eventual correctness if, whenever changes cease for a long enough period, it guarantees all \( f_i \) will converge to \( f(\bigoplus X) \) as computed on the current set of inputs. Often, however, changes never cease for a long enough period, and intermittent accuracy – the percent of peers which compute the correct outcome – is more important than convergence. An algorithm is local if the resources every peer requires in order to arrive at a prescribed level of intermediate accuracy tend to a constant when the number of peers tends to infinity.

Under the said assumptions we denote the agreement of \( p_i \) and its neighbor \( p_j \in N_i \) by \( A_{i,j} = X_{i,j} \oplus X_{j,i} \). We note that unless messages are still traveling between \( p_i \) and \( p_j \), \( A_{i,j} = A_{j,i} \). The state of \( p_i \) is denoted \( S_i = X_i \oplus \bigoplus_{p_j \in N_i} (X_{j,i} \ominus X_{i,j}) \). The function each peer computes, \( f_i \), is defined throughout the paper to be \( f_i = f\left(\sum S_i\right) \). Finally, the function \( f \) considered in this paper is \( f : \mathbb{R}^d \rightarrow \mathcal{R} \cup \{\text{nil}\} \), where \( \mathcal{R} = \{R_1, R_2, \ldots\} \) is a (possibly infinite) set of convex regions in \( \mathbb{R}^d \) and \( \text{nil} \equiv \mathbb{R}^d \setminus \bigcup \mathcal{R} R_i \) their complementary region, and where

\[
  f(\sum X) = \begin{cases} \arg \min \{R_i \in \mathcal{R} : \sum X \in R_i\} & \text{if } \sum X \in \bigcup \mathcal{R} R_i \\ \text{otherwise} \end{cases}
\]

A specific case of interest is where \( \mathcal{R} \) is a convex cover of \( \mathbb{R}^d \).

III. A LOCAL STOPPING RULE FOR GENERAL NETWORK GRAPHS

The correctness of the new algorithm is based on a new stopping rule whose proof does not require that the network be cycle free. We first show that throughout the workings of the algorithm, the average of the inputs is reflected in the states the different peers maintain. This provides that the global input is preserved in the states regardless of how it is distributed by the algorithm.
Theorem 2. [Mass Conservation (Proposition 2.2)] The average of the states of all peers is equal to the average of the inputs of all peers; i.e., \( \sum_{p_i \in P} S_i = \sum_{p_i \in P} X_i \).

Proof: The average of all states is \( \sum_{p_i \in P} S_i = \sum_{p_i \in P} \left( X_i \oplus \left( \sum_{p_j \in N_i} (X_{i,j} \oplus X_{j,i}) \right) \right) \). Rearranging the right-hand side yields \( \left( \sum_{p_i \in P} X_i \right) \oplus \left( \sum_{p_i \in P} \left( \sum_{p_j \in N_i} (X_{i,j} \oplus X_{j,i}) \right) \right) \). Since every \( X_{i,j} \) appears inside the square brackets twice, once preceded by \( \oplus \) and once by \( \ominus \), we have that \( \left( \sum_{p_i \in P} \left( \sum_{p_j \in N_i} (X_{i,j} \oplus X_{j,i}) \right) \right) = 0 \). In other words, it is an identity element. It follows that \( \sum_{p_i \in P} S_i = \sum_{p_i \in P} X_i \).

Next we prove our main theorem, which is a local stopping rule that makes no assumptions on the network graph properties. Local stopping rules are the cornerstone of local thresholding algorithms. The importance of the local stopping rule is that each peer can verify its conditions and then stop communicating, relying on the correctness of the stopping rule to provide that one of two things will happen: Either all peers stop communicating, and hence the local status represents the global average, or there must be a peer which does not stop communicating, and hence computation will continue.

Theorem 3. [Local Stopping Rule] If for all \( p_i \in P \), every \( p_j \in N_i \), and for some convex \( R \subseteq \mathbb{R}^d \), it holds that \( |S_i \ominus A_{i,j}| \neq 0 \Rightarrow S_i \ominus A_{i,j} \in R \) and \( |A_{i,j}| \neq 0 \Rightarrow A_{i,j} \in R \), then \( \bigoplus X \in R \) as well.

Proof: Let \( T = (P, E) \) be a spanning tree over the network graph rooted at an arbitrary peer. Consider a convergecast process in which, starting in the leaves, every peer \( p_i \) waits until it receives a message from all of its descendants and then sends a message to its parent \( p_j \) which increases the value of the agreement \( A_{i,j} \) by \( S_i \).

First consider a leaf \( p_i \). The leaf waits for no incoming messages and sends a message to its parent \( p_j \). To set the new \( A'_{i,j} \) to \( S_i \), the content of the message should be \( S_i \odot X_{i,j} \). This way \( A_{i,j} = S_i \odot X_{i,j} \delta X_{j,i} = S_i \odot A_{j,i} \). Sending the message increases \( X_{i,j} \) by \( S_i \), and this increase is deduced from the current \( S_i \), which results in a new status \( S'_i = S_i \odot S_i = 0 \).

Now, consider the change as it is experienced by \( p_j \). Its agreement with \( p_i \) changes from \( A_{i,j} \) to \( A_{i,j} \odot S_i \). However, following from the triangle inequality and the convexity of \( R \) (see Section 1.2A), it follows from \( A_{i,j} \odot S_i \) and \( S_i \), being in \( R \) that the new agreement \( A'_{i,j} = S_i \odot A_{i,j} \) is in \( R \) as well.

The status of \( p_j \), \( S_j \), also changes as a result of the change in \( X_{i,j} \). Since \( X_{i,j} = X_{i,j} \odot S_i \), the new value of \( S_j \), \( S'_j \), is \( S'_j = S_j \odot S_i \). Again, since \( S_i \) and \( S'_i \) are in \( R \), so is \( S'_j = S_j \odot S_i \). Finally, for any other neighbor of \( p_j \), \( p_k \notin \mathbb{P}_i \), the value of \( S_j \odot A_{j,k} \) is increased by \( S_i \). Since \( S_j \odot A_{j,k} \in R \) and \( S_i \in R \), \( S'_j \odot A_{j,k} = S_i \odot S_j \odot A_{j,k} \in R \).

We conclude that when a leaf \( p_i \) sends a message to its parent \( p_j \), \( S_i \) becomes a zero element of \( \mathbb{W} \) and the conditions of the theorem continue to hold for \( p_i \) and \( p_j \). The same happens when all of the leaves below any peer send their message. Therefore, by induction, the same happens when a non-leaf \( p_i \) receives the final message from any of its descendants and sends the message to its own parent \( p_j \).

Finally, when the root of the spanning tree, call it \( p_0 \), has received all of the messages, we have \( S_i = 0 \) for any \( p_i \neq p_0 \) and \( S_0 \in R \). From Theorem 2 \( \bigoplus X = \bigoplus_{p_i \in P} S_i \) and since in \( \bigoplus_{p_i \in P} S_i = S_0 \oplus S_1 \oplus \ldots \oplus S_n \) the latter \( n-1 \) elements are zero elements, we have that \( \bigoplus X = \bigoplus_{p_i \in P} S_i = S_0 \). Thus, \( \bigoplus X = S_0 \in R \), which proves the theorem.

We stress that the stopping criterion does not require that the graph be a tree. It is important that the reader not be confused by the spanning tree in the proof of the Theorem. Using a spanning tree is a methodology, not a condition for correctness.

IV. BALANCE CORRECTION

A local stopping rule is just one part of a local thresholding algorithm. It must be complemented by a method for achieving the conditions set by the local stopping rule. In this section we first prove that for any set of inputs, any network topology, and any convex region \( R \) containing \( \bigoplus X \), there exists a solution which satisfies the conditions set by Theorem 3. Then, we describe a pair of local correction policies, each of which is a formula by which a peer can compute outgoing messages such that after the messages are sent the conditions hold at that peer.

A. Existence of a solution

We first show that regardless of the topology and the input of the different peers, there is a set of values for the different \( X_{i,j} \) such that all non-zero \( S_i \odot A_{i,j} \) are equal to all non-zero \( A_{i,j} \). This means that they all must reside in the same \( R \), regardless of its shape. Consider any spanning tree of the network graph. The solution is to assign zero weight to \( X_{i,j} \) and \( X_{j,i} \) corresponding to edges not on a spanning tree. The other \( X_{i,j} \) and \( X_{j,i} \) will be assigned values which provide that for every \( p_i \in P \) and \( p_j \in N_i \) the following holds: \( S_i \odot A_{i,j} = A_{i,j} = \frac{1}{2} |S_i| \odot \bigoplus X \).

Theorem 4. [Termination state existence] For any connected network graph and any set of inputs and a convex \( R \) such that \( \bigoplus X \in R \), there is a setup of values for \( X_{i,j} \) such that the conditions of Theorem 3 are met.

Proof: Let \( T = (V, E') \) be a spanning tree over \( G = (V, E) \) and let \( V_i \) be the vertices in \( p_i \)'s subtree. Let the global weighted average be \( \bigoplus X \). For every \( (p_i, p_k) \in E \) such that \( (p_i, p_k) \notin E' \), let \( |X_{i,k}| = |X_{k,i}| = 0 \), which satisfies the conditions of Theorem 3.

Define the subtree status \( Y_i \) recursively as follows: For a leaf node, the subtree status is equal to the input, \( Y_i = X_i \). For a non-leaf node, \( Y_i \) is the status omitting the last message sent to and received from its parent, \( Y_i = X_i \oplus \bigoplus_{p_i \neq p_i \in N_i} X_{i,k} \odot X_{k,i} \).

Now, we define a message which should be sent and one which should be received from each node to any neighbor on the spanning tree. Let \( p_i \) be a node and \( p_j \) its parent on
we have that after the messages are sent, all of
Applying induction, we get that for any $i\rightarrow X$ parents, we have $A$ new values of the respective
has $Y$ with its own parent average of the inputs in
Theorem 3 to hold for $X$ the tree. We set $A$ to
the conditions of Theorem 3 regardless of what
It follows that for every peer $p_i$ and any parent $p_j$ we have $X = X_i \oplus \left( \bigoplus_{p_k \neq \in N_i} \left( X_{i,k} \oplus X_i \right) \right)$ and $X_i = \bigoplus_{p_i \in V_i} \left( X_{i} \oplus X_i \right)$.

It follows that for every peer $p_i$ and any parent or non-parent, we have $A_{i,j} = S_i \oplus A_{i,j} = X_i \oplus X_i$, and thus $A_{i,j} = S_i \oplus A_{i,j} = X_i \oplus X_i$, which agrees with the conditions of Theorem 3 regardless of what $R$ is.

We note, once more, that although the proof of Theorem 4 uses a spanning tree, the correctness of the stopping rule does not rely on the topology. The proof only serves to show that at least one global termination state exists.

However, a peer cannot directly compute this specific termination state unless it has knowledge of the global topology and of all inputs. Therefore, we now move to describe how local correction methods can be directly computed by a peer using only its own state and agreements with neighbors. These methods do not necessarily provide global termination. Rather, they restore the stopping condition at the peer which uses them, while possibly by violating the condition of its neighbor.

### B. Local correction

Consider a peer $p_i$ whose state currently violates the conditions of Theorem 3, i.e., for some of the $p_j \in N_i$ either $A_{i,j} \not\in R$ and $|A_{i,j}| > 0$ or $S_i \oplus A_{i,j} \not\in R$ and $|S_i \oplus A_{i,j}| > 0$. A local correction heuristic is a method for computing messages for some of the peers in $N_i$ such that the new values of the respective $X_{i,j}$'s cause the conditions of Theorem 3 to hold for $p_i$. Since this paper describes a general algorithm, we are interested in local correction methods whose success does not depend on the state of the peer, the topology, and the specific $R$ in question.

A simple way to achieve independence of $R$ is to make sure that after the messages are sent, all $A_{i,j}$ and all $S_i \oplus A_{i,j}$ are equal. This way, if one is inside $R$, then all are inside $\overline{R}$ and the conditions of Theorem 3 are met. First, note that $S_i \oplus A_{i,j} = A_{i,j}$ means $S_i = A_{i,j} \oplus A_{i,j} = 2 \oplus A_{i,j} = A_{i,j}$, i.e., the local correction method must compute new values for the $X_{i,j}$ of $p_j \in N_i$ such that

$$\forall p_j \in N_i : A_{i,j} = \overline{S_i}. \tag{1}$$

**Theorem 5.** [Perfect correction] Let $A_{i,j}'$ and $S_{i}'$ denote the new values computed by changing each $X_{i,j}$ to $X_{i,j}'$.

Then $A_{i,j}' = S_{i}'$ for all $p_j \in N_i$ if and only if $A_{i,j} = |A_{i,j}'| \oplus \left( X_{i} \oplus \bigoplus_{p_k \in N_i} 2 \oplus X_{k,i} \right)$.

**Proof:** Because the different $A_{i,j}'$ are all equal to $A_{i,j}$, we write for all $p_j, p_k \in N_i$:

$$A_{i,j} = A_{i,k}$$

$$X_{i,j} = X_{i,k} \oplus X_{k,i}$$

$$X_{i,k}' = (X_{i,j}' \oplus X_{j,i}) \oplus X_{k,i}$$

$$X_{k,i}' \oplus X_{j,i} = (2 \oplus X_{k,i}) \oplus (X_{i,j}' \oplus X_{j,i})$$

The equation $\forall p_j \in N_i : A_{i,j}' = S_{i}'$ can be rewritten

$$\forall p_j \in N_i : X_{i,j}' \oplus X_{j,i} = X_i \oplus \bigoplus_{p_k \in N_i} (X_{i,k} \oplus X_{k,i})$$

$$= X_i \oplus \bigoplus_{p_k \in N_i} \left[ (2 \oplus X_{k,i}) \oplus (X_{i,j}' \oplus X_{j,i}) \right]$$

$$= X_i \oplus \bigoplus_{p_k \in N_i} (2 \oplus X_{k,i}) \oplus [N_i \cap (X_{i,j}' \oplus X_{j,i})]$$

which is equivalent to stating, $\forall p_j \in N_i$, that:

$$\left| [N_i] + 1 \right| \cap (X_{i,j}' \oplus X_{j,i}) = X_i \oplus \bigoplus_{p_k \in N_i} \left[ 2 \oplus X_{k,i} \right]$$

Since multiplication by a constant only changes the weight, not the vector part of a weighted vector, we have that

$$\forall p_j \in N_i : X_{i,j}' \oplus X_{j,i} = X_i \oplus \bigoplus_{p_k \in N_i} \left[ 2 \oplus X_{k,i} \right] \tag{2}$$

The set of possible $A_{i,j}'$ which satisfy the requirement in Eq. 1 can be computed by normalizing the weighted vector to the desired $|A_{i,j}'|:

$$A_{i,j}' = \frac{|A_{i,j}'|}{X_i \oplus \bigoplus_{p_k \in N_i} \left[ 2 \oplus X_{k,i} \right]} \oplus \left( X_i \oplus \bigoplus_{p_k \in N_i} \left[ 2 \oplus X_{k,i} \right] \right) \tag{3}$$

In other words, what Theorem 5 states is that if the peer chooses a new weight for its agreement with a neighbor, $|A_{i,j}'|$, then that weight dictates the value of $A_{i,j}$ which would satisfy Eq. 1. Since the peer can enforce both $|A_{i,j}'|$ and $A_{i,j}'$ by choosing appropriate $X_{i,j}'$ and $X_{i,j}'$, Theorem 5 identifies a range of outgoing messages that satisfy this Eq. 1 which are those which can be computed using Eq. 3.
Theorem 5 provides the general solution to the problem of computing outgoing messages that will set $p_i$’s state to one which obeys the conditions of the stopping criterion, described in Theorem 3. However, the equation does not dictate a single solution but rather a proportion between the weight of the outgoing message, $|X'_{i,j}|$, and the vector of that message, $\bar{X}_{i,j}$. Each choice of $|X'_{i,j}|$ will have a different effect on the states of the sender and the recipient.

The most important property to be had in a correction scheme is guaranteed convergence to a global solution. From results obtained for other iterative averaging algorithms [14], we know that it is simple to achieve convergence of the $|S_i|$ of all $p_i$ to a constant value that depends only on the topology: each time a peer’s state violates the stopping criterion, it will distribute half of $|S_i|$ to its neighbors. So the total weight to be distributed is given by the equation

$$\sum_{p_j \in N_i} (|X'_{i,j}| - |X_{i,j}|) = \frac{|S_i|}{2}.$$  \hspace{1cm} (4)

However, naive implementation of this policy may lead to very small $|S_i|$ values at some peers. While mathematically small weights do not pose a problem, in practice they would lead to numerical instability. Therefore, a minimal weight $\beta$ is enforced on $|S_i|$ by using $\sum_{p_j \in N_i} (|X'_{i,j}| - |X_{i,j}|) = \frac{|S_i| - \beta}{2}$. We find that a small $\beta$ does not hinder convergence.

1) Uniform weight distribution: A simple weight distribution policy is to allocate a constant portion of $|S_i|$ to each $p_j \in N_i$, i.e., to set $|X'_{i,j}| - |X_{i,j}| = \frac{|S_i| - \beta}{2|N_i|}$. Since $|X'_{i,j}| - |X_{i,j}| = |A'_{i,j}| - |A_{i,j}|$, the new $|A'_{i,j}| = \frac{|S_i| - \beta}{2|N_i|} + |A_{i,j}|$. We denote this the uniform weight distribution method and formally define it by instantiating Eq. 5:

$$A'_{i,j} = \frac{|A_{i,j}| + \frac{|S_i| - \beta}{2|N_i|}}{|X_i + \bigoplus_{p_k \in N_i} 2 \odot X_{k,i}|} \odot \left( \bigoplus_{p_k \in N_i} 2 \odot X_{k,i} \right).$$  \hspace{1cm} (5)

2) Selective local correction: By distributing the weight uniformly, as described in Theorem 5 above, a new value is computed for every $X_{i,j}$. This is often unnecessary as many of the neighbors $p_j \in N_i$ may have $A_{i,j}$ and $S_i \odot A_{i,j}$ that still fall inside $R$. Setting the $X_{i,j}$ of those neighbors to a new value can be doubly wasteful: a message must be sent to every neighbor $p_j$, and the change of $X_{i,j}$ might well change some $S_j \odot A_{j,k}$ at $p_j$ to the degree that it violates the stopping condition, triggering further messages. A solution which selectively sends messages to only part of the neighbor set $N_i$ and still brings all of the $A_{i,j}$ and $S_i \odot A_{i,j}$ into $R$ is therefore desirable.

Denote $V_i$ the set of neighbors for whom the stopping condition is violated, $V_i = \{p_j \in N_i : A_{i,j} \notin R \lor S_i \odot A_{i,j} \notin R\}$. The complementary set, $N_i \setminus V_i$, are the neighbors whose update may be avoided. Consider an imaginary peer $p_{im}$ with $N_{im} = V_i$, $X_{im,j} = X_{i,j}$, and $\bar{X}_{im,j} = \bar{X}_{i,j}$ for all $p_j \in N_i$ and $X_{im} = X_i + \bigoplus_{p_j \in N_i \setminus V_i} X_{j,i} \oplus X_{i,j}$. Note that $X_{im} + \bigoplus_{p_j \in N_{im}} 2 \odot X_{j,im} = S_{im} + \bigoplus_{p_j \in N_{im}} A_{im,j}$.

If $p_{im}$ sets every $X_{im,j}$ according to Eq. 2, then we have

$$A'_{im,j} = \frac{|A'_{im,j}|}{|S_{im} + \bigoplus_{p_k \in N_{im}} A_{im,k}|} \odot \left( S_{im} + \bigoplus_{p_k \in N_{im}} A_{im,k} \right) \hspace{1cm} (6)$$

$$= \frac{|A'_{j}|}{|S_j + \bigoplus_{p_k \in V_i} A_{k,i}|} \odot \left( S_j + \bigoplus_{p_k \in V_i} A_{k,i} \right) \hspace{1cm} (7)$$

$$= |A'_{j}| \odot \left( X_i + \bigoplus_{p_k \in N_{im} \setminus V_i} (X_{k,i} \oplus X_{k,j}) + \bigoplus_{p_k \in V_i} 2 \odot X_{k,i} \right)$$

$$|X_i + \bigoplus_{p_k \in N_{im} \setminus V_i} (X_{k,i} \oplus X_{k,j}) + \bigoplus_{p_k \in V_i} 2 \odot X_{k,i} |.$$  \hspace{1cm} (8)

It then would follow that if for all $p_j \in N_{im}$ (i.e., in $V_i$), $A'_{im,j}$ (which is equal to $A'_{i,j}$) is set according to the policy detailed in Eq. 5, then $|S'_{im}| = |S_i| - \frac{|S_i| - \beta}{2}$. The selective version of the uniform weight distribution policy is therefore:

$$A'_{j} = \frac{|A_{j}| + \frac{|S_i| - \beta}{2|N_i|}}{|S_i + \bigoplus_{p_k \in V_i} A_{k,i}|} \odot \left( S_i + \bigoplus_{p_k \in V_i} A_{k,i} \right).$$  \hspace{1cm} (9)

The problem with a selective policy lies with the neighbors that did not violate the stopping condition, those in $N_i \setminus V_i$. Since setting the $X_{i,j}$ of each $p_k \in V_i$ changes the status $S_i$, it may well be that for $p_j \in N_i \setminus V_i$ the vector $S_i \odot A_{i,j}$ is no longer in $R$. One solution can be to iteratively add neighbors to $V_i$ if they violate the stopping condition, and to terminate the iteration when $V_i$ no longer grows. At this stage only, messages are sent to all of the neighbors in $V_i$. Note that, at worst, iteration ends with $V_i = N_i$, which is the non-selective solution.

V. Source selection

The stopping and correction methods presented in the previous sections are general. They can be applied to various sets of convex regions. In this section we demonstrate their application to the problem of source selection. The source selection problem is a generalization of the majority voting problem in which votes and options are vectors in $\mathbb{R}^d$ rather than the points \{0, 1\}. The generalization is sufficiently rich to allow reduction from data mining problems such as decision tree induction [4] and $k$-median [18], and yet simple enough to allow thorough and application-independent experimentation.

Let $C = \{c_1, c_2, \ldots, c_k \in \mathbb{R}^d\}$ be a set of options and let $X_i$ be the input of $p_i$ such that $|X_i| = 1$ and $\bar{X}_i \in \mathbb{R}^d$. The objective of the peers in the source selection problem is to compute $f \left( \bigoplus \bar{X}_i \right) = \arg \min_{c_i \in C} \left\{ \left| c_i - \bigoplus \bar{X}_i \right| \right\}$, where the norm $||\cdot||$ can be any norm (we use the L2 norm). Note that $\mathcal{R} = \{c_1, c_2, \ldots, c_k\}$, where $c_i = \{x \in \mathbb{R} : f(x) = c_i\}$ is a set of convex regions, and nil, the complementary region, is empty. Thus, we can apply local thresholding to the problem with the new stopping rule and the local correction policies described in Section IV-B.

To solve the source selection problem, peers follow the general setup presented above. They exchange weighted vectors and retain the latest vector sent and the latest received from $1$To see that this problem reduces to a majority vote, consider $\bar{X}_i \in \{0, 1\}$ and $C = \{0, 1\}$. 
Algorithm 1 Local Source Selection in General Network Graphs

Common inputs for all peers: $\beta \in [0, 1]$, $\ell \in \mathbb{N}$, $C = \{c_0, c_1, \ldots, c_k \in \mathbb{R}^d\}$

Private input of $p_i$: $\vec{x}_i \in \mathbb{R}^d$, $N_i \subset V$

Output of $p_i$: $f(\frac{\vec{x}_i}{||\vec{x}_i||})$

Initialization:
$X_i \leftarrow (\vec{x}_i, 1)$, $\ell_i \leftarrow -\ell$, $seq_i \leftarrow 0$
For all $p_j \in N_i$
- $X_{i,j}, X_{j,i} \leftarrow (0, 0)$
- $last_j \leftarrow 0$

On a message $(X, seq)$ from $p_j \in N_i$:
If $seq \geq last_j$ then
- $last_j \leftarrow seq$
- $X_{i,j} \leftarrow X$
- $A_{i,j} \leftarrow X_{i,j} \oplus X_{j,i}$
- $S_i \leftarrow X_i \oplus \bigoplus_{p_j \in N_i} (X_{j,i} \ominus X_{i,j})$

On initialization, on any change to $S_i$, and on timer expiration:
If $currentTime() \leq \ell_i < \ell$
- Set timer to $currentTime() + \ell - \ell_i$ and return $V_i \leftarrow \{p_j \in N_i : f(\frac{A_{i,j}}{||A_{i,j}||}) \neq f(\frac{S_i}{||S_i||}) \vee f(\frac{A_{i,i} \oplus A_{i,j}}{||A_{i,i} \oplus A_{i,j}||}) \neq f(\frac{S_i}{||S_i||})\}$
If $V_i = \emptyset$ return $oldS_i \leftarrow S_i$
Do
- $newS_i \leftarrow oldS_i \oplus \bigoplus_{p_j \in V_i} A_{i,j}$
- $\forall p_j \in V_i \text{ do } X_{i,j} \leftarrow \frac{2V_i}{||newS_i||} \oplus newS_i \ominus X_{i,j}$
- $S_i \leftarrow X_i \oplus \bigoplus_{p_j \in N_i} (X_{j,i} \ominus X_{i,j})$
- $W_i \leftarrow \{p_j \in N_i : f(\frac{A_{i,j}}{||A_{i,j}||}) \neq f(\frac{S_i}{||S_i||}) \vee f(\frac{A_{i,i} \oplus A_{i,j}}{||A_{i,i} \oplus A_{i,j}||}) \neq f(\frac{S_i}{||S_i||})\}$
- $V_i \leftarrow V_i \cup W_i$
While $W_i \neq \emptyset$
$seq_i \leftarrow seq_i + 1$
$\ell_i \leftarrow currentTime()$
For all $p_j \in V_i$ send $(X_{i,j}, seq_i)$ to $p_j$

To make evaluation more specific, it is carried out in the context of the LSS algorithm. However, performance on specific applications and in specific system settings would have to be evaluated per case. To facilitate that, the experiments were carried out with a standard simulator, peersim [24], and the code is available on-line.

A. Experimental setup

LSS performance is influenced by four types of parameters. The first are those of the system: the number of peers, $n$, the topology in which they are arranged, and the message drop rate, $r$. In our experiments we use three different topologies: To investigate unstructured peer-to-peer systems we use the Barabasi-Albert [11] model, which is a well-known approximation of the Internet router topology and which is claimed to approximate the structure of systems like Gnutella [28]. To investigate structured peer-to-peer systems, we use the popular Chord topology [29] with the variant that connection with fingers is assumed bidirectional (in essence, implementing Symmetric Chord [23]). For the third target topology, wireless sensor networks, no standard accepted model exists, and mere connectedness is a design challenge [20]. We therefore opt for a wireless sensor network in which sensors are locations on a bi-dimensional grid. In all those three topologies, the average connectivity $|N_i|$ can be controlled, although in Chord its value is typically much larger ($\log(n)$) than in the other two. We experiment with reliable communication and with a range of message drop rates.

The second type of parameters are those related to the function computed, which in the case of LSS are the number of different sources from which the selection is made, $k$, the dimensionality of the data, $d$, and the distribution of the data. In our experiments, the data is normally and independently distributed along each dimension. We randomly select one of the possible sources as the desired outcome of the algorithm and denote the contender its nearest neighbor. The mean of the data is set to a weighted average of the desired outcome and the contender. The weight given to the contender, between zero and one-half, is denoted the bias of the data. The standard deviation of the data is selected as a multiplier of the distance between the desired outcome and the contender. That multiplier is denoted std. When the data is dynamic, inputs are resampled from the same distribution at every cycle. The proportion of peers whose data changes at every simulation cycle is denoted the noise rate, which is measured in units of changed peers per million simulation cycles (ppmc).

Figure 1 depicts an example of two hundred data points with $d = 2$, $k = 3$. The mean of the data is denoted by the X, which is located at a bias of 40% between the desired outcome (square) and the contender (circle). The standard deviation in this example equals once the distance between the desired outcome and the contender (std = 100%).
The last type of parameters are those which can be controlled by the user: the minimum weight parameter $\beta$ and the leaky bucket parameter $\ell$.

Four sets of experiments were carried out. The first two survey the effect of each parameter using static data, the third examines dynamic data and the last the effect of system dynamics. In each experiment, all but one of the parameters were set to a default value: $n = 10,000$, $|N_i| \approx 4$, a drop rate of zero (i.e., reliable communication), $k = 3$, $d = 2$, a bias of 10% and std of 100%, $\beta = 0.001$ and $\ell = 1$. Then, the simulation was run ten times for each tested value of the remaining parameter.

In all of the experiments in which the data is static, performance is measured by the number of cycles needed for convergence of 95% and 100% of the peers, and by the number of messages required for convergence of all peers (convergence of the last 5% of the peers usually requires very few messages). To allow comparison of different topologies, the average number of messages per edge is reported, rather than the grand total. When the data is dynamic, convergence never occurs and messages never cease. The performance metrics used are, therefore, the average accuracy (percent of peers computing the wrong outcome) and average communication cost (number of messages sent per communication link per cycle). In the literature this last metric is sometimes called normalized messaging and has a maximal value of two in case $\ell = 1$ and $\frac{2}{\ell}$ in general.

**B. System properties**

For every distributed algorithm, and especially one intended for large systems, scalability is the single most important criterion. Figure 1 depicts the scale-up of LSS with selective uniform correction. The number of cycles required for complete quiescence and for convergence of 95% of the peers is depicted on the left (sub-figure 2a), and the number of messages per link is depicted on the right (sub-figure 2b).

The first thing to notice is that LSS overhead seems to converge to a constant as the system is scaled up. This is certainly true for the number of cycles until 95% of the peers converge, and for the communication. Although convergence of 100% percent of the peers is an interesting metric, its value is mainly theoretical. First, it is a worst-case metric that depends on the worst performing peer. Second, the typical working scenario of a large distributed system is dynamic, and does not allow 100% convergence at all.

The second thing to notice is the instability of the performance when the topology is Barabasi-Albert. A deeper look into the results reveals that performance is greatly influenced by outliers: single experiments in which the overhead was exceptionally high. We note that Barabasi-Albert is different from both Chord and Grid topologies in the sense that there is no strict limit on peer connectivity. As can be seen in Figure 5b, Barabasi-Albert is also the most sensitive to average peer connectivity. Since each experiment is carried out using a constant topology, topological effects are not averaged out and may well explain outliers.

Besides scale, the other important property of the system is connectivity, as measured in the average size of $|N_i|$. Because of the inert differences between the three topologies tested, not all were tested on the same range of average $|N_i|$. However, as can be seen in figures 4a and 4b, the effect of increased connectivity on LSS is to expedite convergence and increase the number of messages per connection. Since the increase in communication load per link appears to be linear while the number of required converges quickly to a constant, there seems to be an optimum point, which in this experiment is around $|N_i| = 6$. This could be an important observation because many systems do allow at least limited control of connectivity.

The third and last important feature of the system is message reliability. In Internet based systems, reliable messaging usually costs very little. Even when reliability is not possible, message loss rate is expected to be very low. Wireless sensor networks are drastically different: message loss rates can be expected to be very high even between immediate neighbors and reliable messaging is usually too costly to be used for intensive computation.

Figures 4a and 4b depict the effect of random and independent message loss on the convergence and the message overhead of LSS. As can be seen, limited message loss has no impact on convergence or messaging overhead. This is attributed to the effect of having multiple paths between peers: so long as corrective messages arrive through one of the paths, computation goes on and does converge. In all topologies, once a critical threshold is exceeded, convergence becomes impossible. This can be seen, in Figure 4a, for the Barabasi-Albert topology at a loss rate of more than 1% and for the other topologies at a loss rate higher than 5%. In further experiments with a higher drop rate, Barabasi-Albert topology was always the most sensitive to message loss and grid topology the least sensitive. This supports the hypothesis on the effect multiple paths, because in grid topology every two neighbors are tightly connected through their other neighbors.
Figure 2: Scale-up

Figure 3: Connectivity

Figure 4: Message loss rate
C. Data properties

Next, we investigate the sensitivity of LSS to the difficulty of the problem. From previous research it is known that the performance of local thresholding algorithms depends mainly on the proximity of the average to the decision threshold. As can be seen in Figures 5a and 5b, the communication overhead and the 95% convergence rate decrease super-exponentially with the bias. The 100% convergence rate also decreases with the bias, although perhaps not exponentially.

Increased noise also makes computation more costly. Figures 5c and 5d show that as the standard deviation is increased from a quarter of the distance between the desired outcome and the contender to four times that distance, convergence time grows linearly and the message overhead grows sublinearly.

D. Ineffective parameters

The number of possible solutions, k, and the dimensionality, d, of the data, can also affect performance. However, experiments with k ranging from 3 to 243 revealed no sensitivity of the performance. A similar result was obtained when the dimension of the data was varied from d = 1 through d = 6. We conclude that, once bias and variance are accounted for, neither the number of possible solutions nor the dimensionality of the data has any bearing on performance. We chose not to present these results graphically.

The algorithm’s parameters, the minimal weight allowed for |S|, \( \beta \), and the leaky bucket period, \( \ell \), were also experimented with. In our experiments, the algorithm underperformed when both were zero. However, setting the parameters to larger values than the default had no noticeable effect. Thus, we keep \( \beta = 0.001 \) and \( \ell = 1 \), and refrain from presenting numeric results.

E. Dynamic data

The next set of experiments is, arguably, the most realistic. In these experiments the data at the peers was randomly changed at a controlled rate for 100,000 simulation cycles. This hinders convergence and causes a constant need for further communication. Thus, the average number of peers which compute the wrong outcome and the average number of messages per link per cycle are reported instead of the convergence rate and total number of messages. Because of the large number of events in these experiments, they were executed on networks of just 1000 peers. Also, these experiments were carried out with twice the default bias (20% rather than 10%) and twice the default standard deviation of the data (twice the distance of the desired outcome from the contender rather than just once that distance) in order to increase the effect of every change.

As can be seen in Figures 6a and 6b, up to a noise rate in which one peer’s input changes, on average, at each simulator cycle, the effect of data dynamics is almost only on communication and not on accuracy. Correction is, apparently, fast enough so that the occasional changed input does not propagate an error to a larger part of the network. On the other hand, that same correction does cost in messages. So communication cost does grow about linearly with the noise rate. Then, at about the noise rate which provides for one change in each simulator cycle, the effect of the change on accuracy begins to become noticeable, and the errors accumulate linearly with the noise rate.

The effect of message loss in a dynamic setup is different than it is when the data is static. In a static setup, a peer who does not receive the intended message does not react to correct the wrong output of the sender. In a dynamic setup, however, a peer has many more triggers that will cause it to react, and message loss has only a short-term effect on correctness.

Evidence for this can be seen in Figure 7a and Figure 7b. In these experiments, the loss rate is gradually increased, and the data is dynamically changed at a rate of one thousand peers per million per simulator cycle. As can be seen, the percentage of peers which compute the wrong result is extremely low. This means that the errors induced by message loss hardly accumulate. When 5% of all messages are lost, the error rate is less than half percent. In comparison, in Figure 4a one can see that at this loss rate the error rate at an experiment with static data skyrockets.

The second phenomenon evident in this dynamic setup is that the variance of both the accuracy (Figure 7a) and the communication overhead (Figure 7b) is very large in Barabasi-Albert and in Chord topologies, but not in the Grid topology. Again, our hypothesis is that a greater number of short alternate routes between every two neighbors increases the algorithm’s robustness to message loss.

F. Dynamic network

Finally, the robustness of the LSS algorithm to peer churn is validated. Again, a network of 2,000 peers is simulated for 100,000 cycles with their data changing at a rate of 1,000 ppmc. Additionally, peers drop out of the system at a controlled rate of between zero (no churn) and four ppmc. It is assumed that churn is detected by the peer’s neighbors, which then recalculate their status and correct as needed.

Figures 8a and 8b depict the effect different churn rates have on the average error and the average message load per link. Beside the churn rate, the x-axis also denotes the percentage of peers remaining active at the end of the 100,000 cycles. As can be seen in Figure 8a, the error rate grows notably as more peers churn. However, even when eventuall churn nears 40% of the peers, no more than 1% of them compute the wrong outcome on average. Message overhead increases with churn, probably due to the increased effort needed to correct the mistaken outcome. However, the trend is not very clear and the overhead is very noisy. It is worth noting that the regularity of the bi-dimensional grid is degraded with churn, which may explain why performance is similar in that topology and in the other two topologies even though they are less regular.

VII. RELATED WORK

This paper makes a fundamental contribution to computation in large distributed systems. As such, it relates to other
Figure 5: Problem difficulty

Figure 6: Dynamically changing data
methods of computation in similar networks. We choose to categorize such methods according to the regime used to dictate which messages are sent.

The first category is algorithms that enforce a strict messaging regime. This category includes convergecast-based in-network computation (e.g., those using MapReduce [27]), which have been used for years in small-scale distributed systems. It also includes algorithms such as [19], in which messages flow through the entire system in a strict order. As systems grow larger, global methods lose their appeal. This is mainly because enforcing order, synchronization, and reliability becomes impermissibly costly.

The second category is algorithms that are based on repeated averaging. General results about the convergence of statistics under repeated averaging are known since the 70’s [9]. They were first implemented for function computation in a distributed system in applications such as diffusive load balancing [23], averaging [33], and Kalman filtering [26]. The first relation of repeated averaging to distributed data mining in peer-to-peer system was apparently in the context of the DREAM project [17]. Diffusion has also been shown to allow solving more general optimization problems than merely averaging [25, 10].

Kempe et al., however, were the first to position repeated averaging in the context of gossip algorithms and to provide much needed bounds on convergence speed [16, 5, 14, 2, 12]. Gossip based algorithms were shown to converge with the logarithm of the network size if uniformly random messaging is possible. Otherwise, their convergence rate depends on the eigen-gap of the network graph [6]. Gossip-based algorithms are also simple and robust and have been applied to a large number of problems (see, e.g., [11, 22]). However, since, at base, their convergence rate depends on the random mixing of inputs, gossip algorithms are still extremely wasteful. In a wireless sensor network, where the messaging budget is scarce, it seems inherently wrong to send messages
at random.

Local thresholding defines the last category of algorithms. Unlike gossip-based algorithms, local algorithms are deterministic. They are also far more data dependent. There are ample situations in which a peer running local thresholding will not send any message at all. Comparative testing of local thresholding and gossip algorithms [32] shows that the former are vastly more efficient.

Previous local thresholding algorithms all rely on the cycle freedom of the network. The proof of the stopping condition they use relies on the fact that the latest message received from a neighbor \( p_j \in N_i \) does not depend on the input of \( p_i \) or on inputs which are accounted for in messages received from other \( p_k \in N_i \).

Besides the difficulty of providing a cycle free network, this also meant those algorithms were critically dependent on the reliability of messages to neighbors. If a message is lost on the way from \( p_i \) to \( p_j \), then there is no alternative path in which the inputs represented in that message can propagate the \( p_j \). This dependency has made them even less suitable to wireless sensor network applications. This work is the first to lift the requirement of cycle freedom.

VIII. CONCLUSION AND FURTHER RESEARCH

The new local stopping rule and update methods presented in this paper remove a difficult barrier to the implementation of local thresholding algorithms in some of the most popular distributed environments. It is further hoped that the novel presentation of these algorithms as operating in the field of weighted vectors will simplify future developments. As demonstrated here, local thresholding can be an extremely efficient way to compute complex functions over large distributed networks, regardless of their topological characteristics.

Recent years have seen a lot of focus on applications of local thresholding algorithms and on their stopping rules. This work, and especially the presentation of the problem in terms of weighted vectors, greatly simplifies the argument in favor of certain update policies. We believe some interesting problems lie in the correction policy. Although much important work has been done on expediting convergence of gossip algorithms and diffusive load balancing [13] algorithms using smart update rules, we are not aware of any parallel work on local thresholding algorithms.

The generalization described here still misses one interesting aspect of some of the target systems: that the communication graph is often directed and weighted. In a wireless sensor network, the signal transmitted by a sensor may well be received by a sensor whose signal it cannot receive. The might also have different energy levels, meaning messages are more costly to one than they are to the other. In structured peer-to-peer systems such as Chord, routing from a peer to a peer in its finger table costs just one message whereas routing in the opposite direction can cost \( \log(n) \) messages. In general, expediting convergence by taking into account message delays, and not merely connectivity, may be an important challenge.

REFERENCES

[1] Reka Albert and Albert-Laszlo Barabasi. Statistical mechanics of complex networks. Rev. Mod. Phys., 74(1):47–97, Jan 2002.
[2] F. Benezit, P. Thiran, and M. Vetterli. The Distributed Multiple Voting Problem. IEEE Journal of Selected Topics in Signal Processing, 5(4):791–804, 2011.
[3] K. Bhaduri, K. Das, K. Borne, C. Giannella, T. Mahule, and H. Kargupta. Scalable, Asynchronous, Distributed Eigen Monitoring of Astronomy Data Streams. Stat. Anal. Data Min., 4(3):336–352, 2011.
[4] P. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta. Distributed Decision-Tree Induction in Peer-to-Peer Systems. Stat. Anal. Data Min., 1(2):85–103, June 2008.
[5] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: Design, Analysis and Applications. In Proceedings of INFOCOM 2005, volume 3, pages 1653–1664, 2005.
[6] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. IEEE Transactions on Information Theory, 52(6):2508–2530, 2006.
[7] M. Cao, D. A. Spielman, and E. M. Yeh. Accelerated Gossip Algorithms for Distributed Computation. In 44th Annual Allerton Conference on Communication, Control, and Computation, 2010.
[8] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri. Distributed Kalman Filtering Based on Consentiance Strategies. IEEE Journal on Selected Areas in Communications, 26(4):622 – 633, 2008.
[9] S. Chatterjee and E. Seneta. Towards Consensus: Some Convergence Theorems on Repeated Averaging. Journal of Applied Probability, 14(1):89–97, 1977.
[10] J. Chen and A. H. Sayed. Diffusion adaptation strategies for distributed optimization and learning over networks. Signal Processing, IEEE Transactions on, 60(8):4289–4305, 2012.
[11] A. Chiuso, F. Fagnani, L. Schenato, and S. Zampieri. Gossip algorithms for simultaneous distributed estimation and classification in sensor networks. IEEE Journal of Selected Topics in Signal Processing, 5(4):691–706, 2011.
[12] I. Eyal, I. Keidar, and R. Rom. LiMoSense - Live Monitoring in Dynamic Sensor Networks. In Proceedings of ALGOSENSE, 2011.
[13] Bhaskar Ghosh, S. Muthukrishnan, and Martin H. Schultz. First and Second Order Diffusive Methods for Rapid, Coarse, Distributed Load Balancing (Extended Abstract). In ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pages 72–81, 1996.
[14] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-Based Aggregation in Large Dynamic Networks. ACM Transactions on Computer Systems, 23(3):219–252, August 2005.
[15] S. Kar and J. M. F. Moura. Gossip and Distributed Kalman Filtering: Weak Consensus under Weak Detectability. IEEE Transactions on Signal Processing, 59(4):1766 – 1784, 2011.
[16] D. Kempe, A. Dobra, and J. Gehrke. Gossip-Based Computation of Aggregate Information. In Proceedings of Foundations of Computer Science (FOCS), pages 482–491, 2003.
[17] W. Kowalczyk, M. Jelasity, and A. Eiben. Towards data mining in large and fully distributed peer-to-peer overlay networks. In Proceedings of BNAIC’03, pages 203–210, 2003.
[18] D. Krivitski, A. Schuster, and R. Wolff. A Local Facility Location Algorithm for Sensor Networks. In Proceedings of DCOSS’05, pages 368–375, Marina del Rey, California, June 2005.
[19] Y. A. Le Borgne, S. Raybaud, and G. Bontempi. Distributed principal component analysis for wireless sensor networks. Sensors, 8(8):4821–4850, 2008.
[20] Ji Li, Lachlan L. H Andrew, Chuan Heng Foh, Moshe Zukerman, and Hsiao-Hwa Chen. Connectivity, coverage and placement in wireless sensor networks. Sensors (Basel, Switzerland), 9(10):7664–93, January 2009.
[21] P. Luo, H. Xiong, K. Li, and Z. Shi. Distributed Classification in Peer-to-Peer Networks. In Proceedings of ACM SIGKDD, pages 968–976, 2007.
[22] J. Bucklew M. Rabbat, R. Nowak. Robust Decentralized Source Localization via Averaging. In Proceedings of ICASSP, pages 1057–1060, 2005.
[23] Valentin Mesaros, Bruno Carton, and Peter Van Roy. S-Chord: Using Symmetry to Improve Lookup Efficiency in Chord. In Proceedings of PDPTA, pages 1752–1760, 2003.
[24] A. Montresor and M. Jelasity. PeerSim: A Scalable P2P Simulator. In Proc. of the 9th Int. Conference on Peer-to-Peer (P2P), pages 99–100, 2009.
[25] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. Automatic Control, IEEE Transactions on, 54(1):48–61, 2009.
[26] R. Olfati-Saber. Distributed Kalman Filter with Embedded Consensus Filters. In Proceedings of CDC-ECC, pages 8179 – 8184, 2005.
[27] Spiros Papadimitriou and Jimeng Sun. DisCo: Distributed Co-clustering with Map-Reduce: A Case Study towards Petabyte-Scale End-to-End Mining. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, pages 512–521, Washington, DC, USA, 2008. IEEE Computer Society.

[28] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In Multimedia Computing and Networking (MMCN), San Jose, 2002.

[29] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. Networking, IEEE/ACM Transactions on, 11(1):17–32, feb 2003.

[30] R. Wolff, K. Bhaduri, and H. Kargupta. Local L2 Thresholding Based Data Mining in Peer-to-Peer Systems. In Proceedings of SDM, pages 430–441, 2006.

[31] R. Wolff and A. Schuster. Mining Association Rules in Peer-to-Peer Systems. IEEE Transactions on Systems, Man and Cybernetics - Part B, 34(6):2426–2438, December 2004.

[32] Ran Wolff. Local Thresholding on Distributed Hash Tables. Technical report, University of Haifa, July 2012.

[33] L. Xiao and S. Boyd. Fast linear iterations for distributed averaging. Systems & Control Letters, 53(1):65–78, 2004.