LETTER **Communicated by Marshall Tappen**

Linear Coordinate-Descent Message Passing for Quadratic Optimization

Guoqiang Zhang
g.zhang-1@tudelft.nl
Richard Heusdens
r.heusdens@tudelft.nl
Department of Intelligent Systems, Delft University of Technology, Delft, the Netherlands

In this letter, we propose a new message-passing algorithm for quadratic optimization. The design of the new algorithm is based on linear coordinate descent between neighboring nodes. The updating messages are in a form of linear functions as compared to the min-sum algorithm of which the messages are in a form of quadratic functions. As a result, the linear coordinate-descent (LiCD) algorithm transmits only one parameter per message as opposed to the min-sum algorithm, which transmits two parameters per message. We show that when the quadratic matrix is walk-summable, the LiCD algorithm converges. By taking the LiCD algorithm as a subroutine, we also fix the convergence issue for a general quadratic matrix. The LiCD algorithm works in either a synchronous or asynchronous message-passing manner. Experimental results show that for a general graph with multiple cycles, the LiCD algorithm has comparable convergence speed to the min-sum algorithm, thereby reducing the number of parameters to be transmitted and the computational complexity.

1 Introduction

We consider solving the quadratic optimization problem in a distributed fashion, namely,

$$\min_{x\in\mathbb{R}^n} f(x) = \min_{x\in\mathbb{R}^n}\left(\frac{1}{2}x^\top Jx - h^\top x\right), \tag{1.1}$$

where the quadratic matrix J is symmetric positive definite and x is a real vector in n-dimensional space. Without loss of generality, we assume that J has unit-diagonal. It is known that the optimal solution x^* satisfies the linear equation $Jx^* = h$. By exploiting the sparsity of the quadratic matrix J, we can decompose $f(x)$ in a pairwise fashion according to a graphic model

$G = (V, E)$ (Lauritzen, 1996),

$$f(x) \triangleq \sum_{(i,j)\in E} f_{ij}(x_i, x_j) + \sum_{i\in V} f_i(x_i). \tag{1.2}$$

That is, every variable x_i in the problem is associated with a node $i \in V$ and takes values over the real line $\mathbb{R}$. The variables contribute to the overall function by interacting with their neighboring ones. To compute the optimal solution, the research challenge is how to spread the global information of (J, h) in equation 1.1 over the graph efficiently by exchanging local information between neighboring nodes.

Our motivation for solving equation 1.1 in a distributed manner stems from problems originating from wireless sensor networks. One such example is distributed speech enhancement in wireless microphone networks (Bertrand & Moonen, 2012; Heusdens, Zhang, Hendriks, Zeng, & Kleijn, forthcoming), where solving a quadratic optimization problem is an essential step. In such networks, due to the absence of a central processing point (fusion center), sensor nodes use their own processing ability to locally carry out simple computations and transmit only the required and partially processed data to neighboring nodes. In general, the sensor nodes have limitations on computational capacity, storage space, and transmission power. As a consequence, the applicability of an iterative algorithm in such networks depends on its computational complexity and the number of parameters to be stored or transmitted.

In the literature, the Jacobi algorithm is a natural approach to solving the quadratic problem, equation 1.1 (Bertsekas & Tsitsiklis, 1997). The Jacobi algorithm updates the estimate of x^* synchronously. Due to its simplicity, the convergence condition of the Jacobi algorithm is well established in terms of J. In contrast to the Jacobi algorithm, the Gauss-Seidel algorithm updates the estimate of x^* asynchronously (or sequentially) (Bertsekas & Tsitsiklis, 1997). Unlike the Jacobi algorithm, the Gauss-Seidel algorithm converges to the optimal solution for any symmetric positive-definite matrix J. For the two algorithms, once the node estimate is updated, this estimate is broadcast to all of its neighbors. Broadcast transmission scheme is favorable in sensor networks. But because the information transmitted is general and not edge specific, both the Jacobi and Gauss-Seidel algorithms are known to converge slowly (Bertsekas & Tsitsiklis, 1997). The conjugate gradient algorithm is a more advanced iterative approach for the quadratic problem (Wolfe, 1978). However, the computation of the updating step size requires a distributed gossip algorithm for averaging consensus (Boyd, Ghosh, Prabhakar, & Shah, 2006), which makes the algorithm less practical in wireless sensor networks.

An alternative scheme for solving the quadratic problem, equation 1.1, is by using the framework of probability theory (Lauritzen, 1996). The optimal solution x^* is viewed as the mean value of a random vector $x \in \mathbb{R}^n$ with gaussian distribution

$$p(x) \propto \exp\left(-\frac{1}{2}x^\top Jx + h^\top x\right).$$

As a popular distributed algorithm, the min-sum message passing estimates not only the mean value $x^* = J^{-1}h$ but also the individual variances (Pearl, 1988; Johnson, Malioutov, & Willsky, 2006). For the min-sum algorithm, each node computes and transmits edge-specific information instead of broadcasting some common parameters to all its neighbors. This edge-specific operation helps to spread the global information of (J, h) over the graph more effectively. Due to the fact that each message is a quadratic function, the min-sum algorithm has to transmit two specific parameters for each message.

The convergence condition of the min-sum algorithm has received considerable attention in the past. It is well known that if the graph has a tree structure (i.e., no loops involved), the min-sum algorithm converges to the true mean and individual variances in finite steps (Pearl, 1988; Weiss & Freeman, 2001). The question of convergence for loopy graph models has been proven difficult. Weiss and Freeman (2001) and Rusmevichientong and Roy (2001), have shown that if the min-sum algorithm converges for general graphs, it computes the mean value x^*. In particular, Weiss and Freeman (2001) established a convergence condition where the quadratic matrix J is required to be diagonally dominant.[1] Later, Johnson et al. (2006) discovered a more general convergence condition (Malioutov, Johnson, & Willsky, 2006). They found that if the matrix J is walk-summable, the min-sum algorithm always converges.[2] Recent work by Moallemi and Roy (2009) provided a geometrical meaning to the walk-summability of J. Johnson, Bickson, and Dolev (2009) proposed a double-loop algorithm to compute the optimal solution for a general matrix J, where the min-sum algorithm is used as a subroutine.

We note that we are interested only in the optimal solution x^* of equation 1.1 in our work. One natural question is if an efficient algorithm exists that combines the advantages of the Jacobi, Gauss-Seidel, and min-sum algorithms. That is, each node transmits only one specialized parameter to each of its neighbors, while the convergence speed is comparable to that of the min-sum algorithm in general. To achieve this goal, the messages of the new algorithm have to be in a form of linear functions. Linear-functional

[1]The matrix J is diagonally dominant if $|J_{ii}| > \sum_{j\neq i} |J_{ij}|$ for all i.

[2]See section 3.1 for the definition of walk-summability. It can be shown by algebra that if a matrix is diagonal dominant, then it is also walk-summable, the converse not always being true.

messages have significant importance to sensor network–related problems, where computation capacity and transmission power are limited.

In this letter, we design a new algorithm with linear-functional messages for the quadratic problem. The messages are updated by performing linear coordinate-descent (LiCD) between neighboring nodes in the graph. The design of the LiCD algorithm is inspired by the block coordinate-descent (BCD) algorithms developed for discrete graph models (Globerson & Jaakkola, 2008; Sontag, Globerson, & Jaakkola, 2011). While BCD algorithms require that the messages are updated on a block basis (block by block), the messages of the LiCD algorithm, can be updated either synchronously or on a block basis.

We will show that when the matrix J is walk-summable, the LiCD algorithm converges to the optimal solution. We emphasize that the walk-summability of J is only a sufficient condition for the algorithm convergence. When the matrix J is not walk-summable, we design a double-loop algorithm by following the work of Johnson et al. (2009). Unlike the algorithm in Johnson et al. (2009), we take the LiCD algorithm as a subroutine in the double-loop algorithm.

We conduct several experiments with synthetic data to evaluate the performance of the LiCD algorithm. Also, we consider the application of the LiCD algorithm to the distributed speech enhancment in wireless microphone networks. We implement the Jacobi and min-sum algorithms for performance comparison. Experiments show that for a general graph with multicycles, the LiCD algorithm is comparable to the min-sum algorithm with regard to convergence speed but has a lower computational complexity and fewer transmission parameters. Also we find empirically that for an arbitrary graph, the LiCD algorithm converges significantly faster than the Jacobi algorithm, even though the two algorithms transmit the same number of parameters per iteration.

The reminder of the letter is organized as follows. In section 2, we present the LiCD algorithm for the synchronous message-passing scenario. In section 3, we establish the convergence of the LiCD algorithm for both the synchronous and asynchronous message-passing scenarios. Section 4 addresses the convergence issue of the LiCD algorithm for a general matrix J. In section 5 and 6, we present the experimental results in detail. Finally we draw conclusions in section 7.

2 Linear Coordinate-Descent Algorithm

In this section, we present the LiCD algorithm in detail for the synchronous message-passing scenario. We note that in general, if an algorithm works for the synchronous scenario, it also works for the asynchronous scenario (e.g., the min-sum and Jacobi algorithms). We consider the LiCD algorithm for the asynchronous scenario in section 3.2.

We first describe the general framework for synchronous message-passing algorithms. We then design the LiCD algorithm within this framework by deriving the updating expressions of the linear-functional messages. We also consider parameter transmissions between neighboring nodes to update the messages. After that, we analyze the computational complexity of the LiCD algorithm. Finally, we consider algorithm implementation.

2.1 Synchronous Message-Passing Framework. Consider the quadratic objective function, equation 1.1. We construct the local functions to be

$$f_{ij}(x_i, x_j) = J_{ij}x_i x_j \quad (i, j) \in E, \tag{2.1}$$

$$f_i(x_i) = \frac{1}{2}x_i^2 - h_i x_i \quad i \in V. \tag{2.2}$$

The local functions f_i and f_{ij} are often called self-potentials and edge potentials, respectively. f_{ij} captures the interaction between nodes i and j. For the quadratic problem, an edge between nodes i and j exists in the graph only if $J_{ij} = J_{ji} \neq 0$. We use $N(i)$ to denote the set of all neighbors of node i. The set $N(i)\backslash j$ excludes the node j from $N(i)$. We denote the degree of node i as d_i, where $d_i = |N(i)|$. For each edge $(i, j) \in E$, we use $[j, i]$ and $[i, j]$ to denote its two directed edges. Correspondingly, we use $\vec{E}$ to denote the set of all directed edges.

A message-passing algorithm exchanges information between neighboring nodes iteratively until reaching consensus. In particular, at time t, each node i collects a set of messages $\{m_{u\to i}^{(t)}(x_i)|u \in N(i)\}$ by cooperating with its neighbors. We note that for a directed edge $[u, i] \in \vec{E}$, the associated message $m_{u\to i}^{(t)}(x_i)$ is a function of x_i obtained by combining the local information of node u and i at time $t-1$. Given the messages at time t, one can define new self and edge potentials at time t as

$$f_{ij}^{(t)}(x_i, x_j) = J_{ij}x_i x_j - m_{j\to i}^{(t)}(x_i) - m_{i\to j}^{(t)}(x_j) \quad (i, j) \in E, \tag{2.3}$$

$$f_i^{(t)}(x_i) = \frac{1}{2}x_i^2 - h_i x_i + \sum_{u\in N(i)} m_{u\to i}^{(t)}(x_i) \quad i \in V. \tag{2.4}$$

It is straightforward that

$$f(x) = \sum_{(i,j)\in E} f_{ij}^{(t)}(x_i, x_j) + \sum_{i\in V} f_i^{(t)}(x_i). \tag{2.5}$$

Thus, the overall objective function remains the same. The new potentials, equations 2.3 and 2.4 can be viewed as a reformulation of the objective function. The Jacobi and min-sum algorithms fit into the above function reformulation of equations 2.3 and 2.4. The messages of the min-sum algorithm are in the form of quadratic functions, while the messages of Jacobi algorithms are in the form of linear functions (see section 2.3 for a detailed analysis). The key part of a message-passing algorithm is the derivation of the updating expressions for $\{m_{j\to i}^{(t+1)}(x_i), [j, i] \in \vec{E}\}$, given the messages at time t.

At time t, each node i computes an estimate $\hat{x}_i^{(t)}$ of the optimal solution x_i^* by minimizing the self-potential:

$$\hat{x}_i^{(t)} = \arg\min_{x_i} f_i^{(t)}(x_i) \quad i \in V. \tag{2.6}$$

If the algorithm converges to the optimal solution x^*, we have

$$\lim_{t\to\infty} \hat{x}^{(t)} = x^*,$$

where $\hat{x}^{(t)} = (\hat{x}_i^{(t)}, \ldots, \hat{x}_{|V|}^{(t)})^\top$ is the estimation vector.

2.2 Updating Expressions and Parameter Transmissions. The LiCD algorithm performs pairwise minimization between neighboring nodes in computing new messages. In this subsection, we first compute the updating expressions for the messages. After that, we consider parameter transmission between neighboring nodes to update the messages.

In order to derive the updating expressions for the quadratic problem, we first specify the functional forms of the messages and the potential functions. Suppose that the messages $\{m_{j\to i}^{(t)}(x_i), [j, i] \in \vec{E}\}$ at time t take a linear form, that is,

$$m_{j\to i}^{(t)}(x_i) = -z_{ji}^{(t)} x_i \quad [j, i] \in \vec{E}. \tag{2.7}$$

By inserting equation 2.7 into equations 2.3 and 2.4, we obtain the potential functions at time t as

$$f_{ij}^{(t)}(x_i, x_j) = J_{ij} x_i x_j + z_{ji}^{(t)} x_i + z_{ij}^{(t)} x_j \quad (i, j) \in E, \tag{2.8}$$

$$f_i^{(t)}(x_i) = \frac{1}{2} x_i^2 - \left(h_i + \sum_{u \in N(i)} z_{ui}^{(t)} \right) x_i \quad i \in V. \tag{2.9}$$

Note that the potential functions involve the change of only the linear terms instead of the quadratic terms, which is fundamentally different from that of the min-sum algorithm.

Without loss generality, we focus on deriving the expressions for $z_{ji}^{(t+1)}$ and $z_{ij}^{(t+1)}$, $(i, j) \in E$, given the potential functions at time t. Before going into detail, we present the basic idea of linear coordinate descent. Note that only three potential functions involve $z_{ij}^{(t)}$ or $z_{ji}^{(t)}$ (or both), which are $f_i^{(t)}(x_i)$, $f_j^{(t)}(x_j)$, and $f_{ij}^{(t)}(x_i, x_j)$. We build a joint function $L_{ij}^{(t)}(x_i, x_j)$ from the three potential functions in computing $z_{ij}^{(t+1)}$ and $z_{ij}^{(t+1)}$. In particular, we define the joint function to be

$$L_{ij}^{(t)}(x_i, x_j) = f_i^{(t)}(x_i) + f_j^{(t)}(x_j) + f_{ij}^{(t)}(x_i, x_j).$$

This particular form of the joint function is motivated by the fact that the expression for $L_{ij}^{(t)}(x_i, x_j)$ is a sub-summation of equation 2.5 for the objective function $f(x)$. As a result, $L_{ij}^{(t)}(x_i, x_j)$ no longer involves $z_{ij}^{(t)}$ and $z_{ji}^{(t)}$. In the block coordinate-descent algorithms (Sontag et al., 2011; Globerson & Jaakkola, 2008), a similar construction for a joint function was used.

In the computation of $\{z_{ij}^{(t+1)}, z_{ji}^{(t+1)}\}$, we first minimize $L_{ij}^{(t)}(x_i, x_j)$ over $\{x_i, x_j\}$. Denote the resulting optimal solution as $\{\hat{x}_{i|j}^{(t+1)}, \hat{x}_{j|i}^{(t+1)}\}$, which is in fact the estimate of $\{x_i^*, x_j^*\}$. The subscript $i|j$ (or $j|i$) indicates that the estimate $\hat{x}_{i|j}^{(t+1)}$ (or $\hat{x}_{j|i}^{(t+1)}$) is computed by utilizing the information from node j (or node i). After obtaining $\{\hat{x}_{i|j}^{(t+1)}, \hat{x}_{j|i}^{(t+1)}\}$, we then choose $z_{ij}^{(t+1)}$ and $z_{ji}^{(t+1)}$ such that

$$\hat{x}_{i|j}^{(t+1)} = \arg\min_{x_i} \left(\frac{1}{2}x_i^2 - \left(h_i + \sum_{u \in N(i) \setminus j} z_{ui}^{(t)} + z_{ji}^{(t+1)} \right) x_i \right), \tag{2.10}$$

$$\hat{x}_{j|i}^{(t+1)} = \arg\min_{x_j} \left(\frac{1}{2}x_j^2 - \left(h_j + \sum_{v \in N(j) \setminus i} z_{vj}^{(t)} + z_{ij}^{(t+1)} \right) x_j \right). \tag{2.11}$$

Compared to equation 2.9, the right-hand sides of equations 2.10 and 2.11 update the messages associated with the edge (i, j). Correspondingly, the estimate $\hat{x}_{i|j}^{(t+1)}$ is different from $\hat{x}_i^{(t)}$ in equation 2.6. The parameter $z_{ji}^{(t+1)}$ brings sufficient information about $\hat{x}_{i|j}^{(t+1)}$ that is contained in node j to node i. The algorithm converges if for every $i \in V$, all the estimates $\{\hat{x}_i^{(t)}, \hat{x}_{i|u}^{(t)}, u \in N(i)\}$ converge to the same optimal solution x_i^*. Since we operate only on the linear terms of the potential functions, this is how the term *linear coordinate descent* arose.

Based on these computation guidelines we now compute $\{\hat{x}_{i|j}^{(t+1)}, \hat{x}_{j|i}^{(t+1)}\}$. From equations 2.8 and 2.9, the expression for $L_{ij}^{(t)}(x_i, x_j)$ is given by

$$L_{ij}^{(t)}(x_i, x_j) = \frac{1}{2}\begin{pmatrix} x_i & x_j \end{pmatrix}\begin{bmatrix} 1 & J_{ij} \\ J_{ij} & 1 \end{bmatrix}\begin{pmatrix} x_i \\ x_j \end{pmatrix} - \left(h_i + \sum_{u \in N(i)\setminus j} z_{ui}^{(t)} \;\; h_j + \sum_{v \in N(j)\setminus i} z_{vj}^{(t)} \right)\begin{pmatrix} x_i \\ x_j \end{pmatrix}. \tag{2.12}$$

Because the 2×2 quadratic matrix of $L_{ij}^{(t)}(x_i, x_j)$ is positive definite, the optimal solution $\{\hat{x}_{i|j}^{(t+1)}, \hat{x}_{j|i}^{(t+1)}\}$ is finite. Minimizing $L_{ij}^{(t)}(x_i, x_j)$ produces

$$\hat{x}_{i|j}^{(t+1)} = \frac{h_i + \sum_{u \in N(i)\setminus j} z_{ui}^{(t)} - J_{ij}\left(h_j + \sum_{v \in N(j)\setminus i} z_{vj}^{(t)}\right)}{1 - J_{ij}^2}, \tag{2.13}$$

$$\hat{x}_{j|i}^{(t+1)} = \frac{h_j + \sum_{v \in N(j)\setminus i} z_{vj}^{(t)} - J_{ij}\left(h_i + \sum_{v \in N(i)\setminus j} z_{ui}^{(t)}\right)}{1 - J_{ij}^2}. \tag{2.14}$$

The optimal solution is independent of $z_{ij}^{(t)}$ and $z_{ji}^{(t)}$.

Given the optimal solution $\{\hat{x}_{i|j}^{(t+1)}, \hat{x}_{j|i}^{(t+1)}\}$, the expressions for $z_{ij}^{(t+1)}$ and $z_{ji}^{(t+1)}$ can then be derived by combining equations 2.10 and 2.11 and equations 2.13 and 2.14:

$$z_{ji}^{(t+1)} = \frac{J_{ij}^2}{1 - J_{ij}^2}\left(h_i + \sum_{u \in N(i)\setminus j} z_{ui}^{(t)}\right) - \frac{J_{ij}}{1 - J_{ij}^2}\left(h_j + \sum_{v \in N(j)\setminus i} z_{vj}^{(t)}\right), \tag{2.15}$$

$$z_{ij}^{(t+1)} = \frac{J_{ij}^2}{1 - J_{ij}^2}\left(h_j + \sum_{v \in N(j)\setminus i} z_{vj}^{(t)}\right) - \frac{J_{ij}}{1 - J_{ij}^2}\left(h_i + \sum_{u \in N(i)\setminus j} z_{ui}^{(t)}\right). \tag{2.16}$$

We later explain the geometrical meaning of $J_{ij}^2/(1 - J_{ij}^2)$ and $-J_{ij}/(1 - J_{ij}^2)$. The updating expressions for other parameters $\{z_{kl}^{(t+1)}, z_{lk}^{(t+1)}, (k, l) \in E\}$ take similar forms as in equations 2.15 and 2.16. The information flow for computing $z_{ij}^{(t+1)}$ and $z_{ji}^{(t+1)}$ is illustrated in Figure 1.

Next we consider parameter transmission between neighboring nodes to update the messages for the LiCD algorithm. Again, without loss of generality, we consider the edge $(i, j) \in E$. Expressions 2.15 and 2.16 for $z_{ij}^{(t+1)}$ and $z_{ji}^{(t+1)}$ share some common terms. We reformulate the expressions

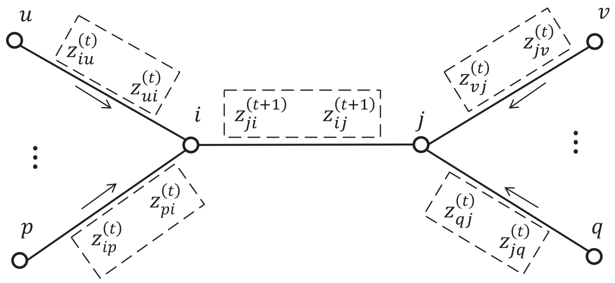

Figure 1: The information-flow for computing $z_{ji}^{(t+1)}$ and $z_{ij}^{(t+1)}$.

as

$$z_{ji}^{(t+1)} = J_{ij}\beta_{i\backslash j}^{(t)} - \beta_{j\backslash i}^{(t)} \tag{2.17}$$

$$z_{ij}^{(t+1)} = J_{ij}\beta_{j\backslash i}^{(t)} - \beta_{i\backslash j}^{(t)}, \tag{2.18}$$

where

$$\beta_{i\backslash j}^{(t)} = \frac{J_{ij}}{1-J_{ij}^2}\left(h_i + \sum_{u\in N(i)\backslash j} z_{ui}^{(t)}\right), \tag{2.19}$$

$$\beta_{j\backslash i}^{(t)} = \frac{J_{ij}}{1-J_{ij}^2}\left(h_j + \sum_{v\in N(j)\backslash i} z_{vj}^{(t)}\right). \tag{2.20}$$

We note that the two parameters $\beta_{i\backslash j}^{(t)}$ and $\beta_{j\backslash i}^{(t)}$ can be computed alone by node i and j, respectively. Thus, in order for node i to update z_{ji} at time t, node j needs to send the parameter $\beta_{j\backslash i}^{(t)}$ only to node i. The other parameters $\{z_{uv}, [u, v] \in \vec{E}\}$ can be updated in a similar way.

To briefly summarize, as the LiCD algorithm evolves, each node $i \in V$ keeps track of the most recent parameters $\{z_{ui}, u \in N(i)\}$. In order to update the parameters $\{z_{uv}, [u, v] \in \vec{E}\}$, each node $i \in V$ computes and transmits $\beta_{i\backslash u}$ to each neighboring node u, $u \in N(i)$. Finally, each node $i \in V$ updates $\{z_{ui}, u \in N(i)\}$ by using its own computed parameters $\{\beta_{i\backslash u}, u \in N(i)\}$ and received ones $\{\beta_{u\backslash i}, u \in N(i)\}$ from neighbors.

2.3 Computational Complexity. In this section, we study the computational complexity of the LiCD algorithm. In addition, we briefly analyze the computational complexities of the Jacobi and min-sum algorithms for comparison.

We note from equations 2.17–2.20 that the computational complexity of the LiCD algorithm is node specific. A node with a larger degree (number of neighbors) requires more computations. Without loss of generality, we consider node j in the graph. The computation of new parameters for time $t+1$ at node j is summarized as

$$\begin{cases} z_j^{(t)} = \sum_{v \in N(j)} z_{vj}^{(t)} + h_j \\ \beta_{j\backslash i}^{(t)} = \frac{J_{ij}}{1-J_{ij}^2}\left(z_j^{(t)} - z_{ij}^{(t)}\right) & i \in N(j) \\ z_{ij}^{(t+1)} = J_{ij}\beta_{j\backslash i}^{(t)} - \beta_{i\backslash j}^{(t)} & i \in N(j) \end{cases}, \tag{2.21}$$

where we assume that $\{\beta_{i\backslash j}^{(t)}, i \in N(j)\}$ are directly received from neighbors. As the factor $\frac{J_{ij}}{1-J_{ij}^2}$ is constant, it can be stored beforehand. The computation for $z_j^{(t)}$ requires $d_j + 1$ additions. The computation for $\{\beta_{j\backslash i}^{(t)}, i \in N(j)\}$ requires d_j additions and d_j multiplications. Finally, the computation for $\{z_{ij}^{(t+1)}, i \in N(j)\}$ requires d_j multiplications and d_j additions. In total, node j performs $3d_j + 1$ additions and $2d_j$ multiplications at each iteration.

Next we consider the Jacobi algorithm. At each iteration, the Jacobi algorithm estimates the optimal solution x^* directly. Denote the estimate vector at time t as $\check{x}^{(t)}$. The algorithm computes the new estimate $\check{x}^{(t+1)}$ by (Bertsekas & Tsitsiklis, 1997),

$$\check{x}^{(t+1)} = R\check{x}^{(t)} + h, \tag{2.22}$$

where $R = I - J$ (I is the identity matrix). With equation 2.22, the updating expression for each component $\check{x}_j^{(t+1)}$, $j \in V$, can be expressed as

$$\check{x}_j^{(t+1)} = h_j - \sum_{i \in N(j)} J_{ij}\check{x}_i^{(t)}. \tag{2.23}$$

With equations 2.22 and 2.23, one can easily work out the message form of the Jacobi algorithm, that is, for $[i, j] \in \vec{E}$ at time t, $m_{i\to j}^{(t)}(x_j) = -J_{ij}\check{x}_i^{(t)}x_j$. Once $\check{x}_j^{(t+1)}$ is computed, node j broadcasts $\check{x}_j^{(t+1)}$ to all its neighbors. We note that for the LiCD algorithm, node j transmits a particular parameter $\beta_{j\backslash i}$ to each neighbor $i \in N(j)$, thus speeding up the information flow over the graph. Considering the computational complexity of the Jacobi algorithm, it is immediate from equation 2.23 that node j performs $d_j + 1$ additions and d_j multiplications in computing $\check{x}_j^{(t+1)}$.

We now consider the min-sum algorithm. Unlike the LiCD and Jacobi algorithms, which transmit only one parameter over each directed edge at each iteration, the min-sum algorithm transmits two parameters over each directed edge. Specifically, the message $m_{j\to i}^{(t)}(x_i)$ at time t takes a quadratic form (Moallemi & Roy, 2009),

$$m_{j\to i}^{(t)}(x_i) = -\frac{1}{2}\mu_{ji}^{(t)}x_i^2 + \tilde{z}_{ji}^{(t)}x_i, \tag{2.24}$$

where $\{\mu_{ji}^{(t)}, \tilde{z}_{ji}^{(t)}\}$ are the two parameters to be transmitted from node j to i. The updating expressions for $\{\mu_{ji}^{(t+1)}, \tilde{z}_{ji}^{(t+1)}\}$ are expressed as (Moallemi & Roy, 2009)

$$\mu_{ji}^{(t+1)} = \frac{J_{ij}^2}{1-\sum_{v\in N(j)\setminus i}\mu_{vj}^{(t)}}, \tag{2.25}$$

$$\tilde{z}_{ji}^{(t+1)} = \frac{J_{ij}}{1-\sum_{v\in N(j)\setminus i}\mu_{vj}^{(t)}}\left(h_j - \sum_{v\in N(j)\setminus i}\tilde{z}_{vj}^{(t)}\right). \tag{2.26}$$

With equations 2.25 and 2.26, the computation of the new parameters for time $t+1$ at node j is summarized as

$$\begin{cases} \mu_j^{(t)} = 1 - \sum_{v\in N(j)}\mu_{vj}^{(t)} & \\ \tilde{z}_j^{(t)} = h_j - \sum_{v\in N(j)}\tilde{z}_{vj}^{(t)} & \\ \nu_{ji}^{(t)} = \dfrac{J_{ij}}{\mu_j^{(t)}+\mu_{ij}^{(t)}} & i\in V(j) \\ \mu_{ji}^{(t+1)} = J_{ij}\nu_{ji}^{(t)} & i\in V(j) \\ \tilde{z}_{ji}^{(t+1)} = \nu_{ji}^{(t)}(\tilde{z}_j^{(t)}+\tilde{z}_{ij}^{(t)}) & i\in V(j) \end{cases}.$$

The computation of $\mu_j^{(t)}$ and $\tilde{z}_j^{(t)}$ requires $2d_j+2$ additions. The computation of $\{\nu_{ji}^{(t)}, i\in V\}$ requires d_j additions and d_j divisions. The computation of $\{\mu_{ji}^{(t+1)}, i\in V\}$ requries d_j multiplications. Finally, the computation of $\{\tilde{z}_{ji}^{(t+1)}, i\in V\}$ requires d_j additions and d_j multiplications. In total, node j performs $4d_j+2$ additions, $2d_j$ multiplications, and d_j divisions. We point out that in practice, a division is more expensive than a multiplication.

Table 1: LiCD Message-Passing for Computing $x^* = \arg\min_x \frac{1}{2}x^T Jx - h^T x$.

	Stage	Operation
1	*Initialize*	$z_{ij}^{(0)} \in \mathbb{R}, \forall [i, j] \in \vec{E}$
2	*Iterate*	For all $[i, j] \in \vec{E}$ Update z_{ij} using equation 2.18. End
3	*Check*	If $\{\hat{x}_i\}$ or $\{\epsilon^{(t)}\}$ have converged, go to stage 4; else, return to stage 2.
4	*Output*	Return $\hat{x}_i, \forall i$.

From a computational point of view, it is now clear that for each iteration, the min-sum algorithm is the most expensive for implementation. The Jacobi algorithm is the cheapest of the three algorithms. From the transmission point of view, both the Jacobi and LiCD algorithms transmit only one parameter per directed edge, while the min-sum algorithm has to transmit two parameters. On the other hand, the Jacobi algorithm performs a broadcast transmission scheme, while the LiCD and min-sum algorithms have to perform edge-specific transmission scheme. The remaining work is to find the convergence rates of the three algorithms. We demonstrate by experiments in section 5 and 6 that the LiCD and min-sum algorithms have comparable convergence rates for a general graph with multiple cycles. Also, the LiCD algorithm significantly outperforms the Jacobi algorithm with regard to convergence speed.

2.4 Algorithm Implementation. In implementating of the LiCD algorithm, we let $z_{ji}^{(0)} \in \mathbb{R}$ for all $[j, i] \in \vec{E}$. Because the algorithm evolves according to equations 2.17 to 2.20) over time, each node $i \in V$ keeps track of the most recent parameters $\{z_{ui}, u \in N(i)\}$. At each time, a node computes an estimate for its variable by applying equation 2.6:

$$\hat{x}_i^{(t)} = h_i + \sum_{u \in N(i)} z_{ui}^{(t)} \quad \forall i \in V, t = 0, 1, \ldots \tag{2.27}$$

If the algorithm converges, the parameters $\{z_{ij}\}$ and the estimates $\{\hat{x}_i\}$ would be stable after some time. Thus, one can terminate the iteration by examining $\{z_{ij}\}$ or $\{\hat{x}_i\}$, or both. For the experimental evaluation of the algorithm, one can alternatively define a termination criterion by involving $\hat{x}^{(t)}$ and x^*. In particular, one can check the error $\epsilon^{(t)} = \|\hat{x}^{(t)} - x^*\|_\infty$ at the end of iteration t, $t = 1, 2, \ldots$. We briefly summarize the algorithm in Table 1.

3 Convergence of the LiCD Algorithm

In this section, we first study the convergence of the LiCD algorithm for the synchronous message-updating scenario as discussed in section 2. In particular, we derive a sufficient condition for the algorithm convergence. With the result for the synchronous message-updating scenario, we then consider the asynchronous convergence of the LiCD algorithm.

3.1 Synchronous Convergence. The walk summability of J was originally discovered as a sufficient condition for the convergence of the min-sum algorithm (Johnson et al., 2006; Malioutov et al., 2006; Moallemi & Roy, 2009). We show in the following that the walk summability of J is also a sufficient condition for the LiCD algorithm to converge. For completeness, we provide the definition of walk summability.

Definition 1 *(Johnson et al., 2006; Malioutov et al., 2006). A positive-definite matrix $J \in \mathbb{R}^{n\times n}$, with all ones on its diagonal, is walk-summable if the spectral radius of the matrix $\bar{R}$, where $R = I - J$ and $\bar{R} = [|R_{ij}|]_{i,j=1}^{n}$, is less than 1 (i.e., $\rho(\bar{R}) < 1$).*

Next we describe how the walk summability of J is connected to the optimal solution of the quadratic problem (see Johnson et al., 2006, and Malioutov et al., 2006, for detailed information). First, note that the optimal solution for the quadratic optimization problem 1.1 is given by

$$x^* = J^{-1}h = (I - R)^{-1}h.$$

If we assume that the matrix $\bar{R}$ has a spectral radius less than 1, then the spectral radius of R is also less than 1 (i.e., $\rho(R) < 1$). Under the condition $\rho(R) < 1$, we express the solution x^* by the infinite series

$$x^* = \sum_{t=0}^{\infty} R^t h, \tag{3.1}$$

where the components of R are $R_{ii} = 0$ and $R_{ij} = -J_{ij}$, if $i \neq j$. The condition $\rho(\bar{R}) < 1$ guarantees that the series $\sum_{t=0}^{\infty} R^t$ is absolutely convergent. In other words, the convergence of $\sum_{t=0}^{\infty} R^t$ does not depend on how its components are arranged in the summation.

With equation 3.1 at hand, each component x_i^* can be expressed as

$$x_i^* = \sum_{t=0}^{\infty} [R^t h]_i \quad i \in V. \tag{3.2}$$

For a particular $t \geq 1$ in equation 3.2, the component $[R^t h]_i$ can be further expressed as a summation of some elementary quantities where each quantity is a product of t elements of R and one element of h. Intuitively such an elementary quantity can be associated with a walk (or path) on the graph.

Formally, we define a walk of length k on the graph to be a sequence of nodes $w = [w_0, \ldots, w_k]$ such that $[w_i, w_{i+1}] \in \vec{E}$, for all $0 \leq i < k$. We refer to w_k and $[w_{k-1}, w_k]$ in $w = [w_0, \ldots, w_k]$ as the end node and end edge, respectively. Given a walk w of length $|w|$, we define a weight of w by the product $\gamma(w) = h_{w_0} R_{w_0 w_1} \cdots R_{w_{|w|-1} w_{|w|}}$. For the special case that $|w| = 0$, we define $\gamma(w) = h_{w_0}$. Given a set $\mathcal{W}_i$ of walks with the common end node i, we define a weight of the set to be

$$\gamma\left(\mathcal{W}_i\right) = \sum_{w \in \mathcal{W}_i} \gamma(w).$$

To simplify the analysis, we let $\bar{\gamma}(\mathcal{W}_i) = \sum_{w \in \mathcal{W}_i} |\gamma(w)|$. That is, $\bar{\gamma}\left(\mathcal{W}_i\right)$ represents the summation of absolute weights for a set. From equation 3.2, it is obvious that $\gamma\left(\mathcal{W}_i\right)$ is an estimate of x_i^*, that is, $\hat{x}_i = \gamma\left(\mathcal{W}_i\right)$. Define $\mathcal{W}_i^*$ to be the (infinite) set of all walks terminating at node i. Similarly, define $\mathcal{W}_{ui}^*$ as the (infinite) set of all walks with the end edge $[u, i]$. Under condition $\rho(\bar{R}) < 1$, we have

$$x_i^* = \gamma\left(\mathcal{W}_i^*\right) = h_i + \sum_{u \in N(i)} \gamma\left(\mathcal{W}_{ui}^*\right).$$

Upon introducing the properties of the walk summability of J, we are ready to study the convergence of the LiCD algorithm. In the following, we first establish the algorithm convergence for the special initialization $\{z_{ij}^{(0)} = 0\}$. With the convergence result for the special initialization, we then consider a general initialization for the algorithm.

3.1.1 Initialization $\{z_{ij}^{(0)} = 0\}$. Under the initialization $\{z_{ij}^{(0)} = 0\}$, we use the walk-sum concept to establish the equivalence between equations {2.15–2.16, 2.27} and equations 3.1 and 3.2. More specifically, we first interpret the parameter z_{ji} as the weight of walks that have the same end edge $[j, i]$. We then show that when $\rho(\bar{R}) < 1$, there is

$$\gamma\left(\mathcal{W}_i^*\right) = h_i + \sum_{u \in N(i)} \gamma\left(\mathcal{W}_{ui}^*\right) = h_i + \sum_{u \in N(i)} z_{ui}^{(\infty)}. \tag{3.3}$$

Consider the updating expression 2.15 for $z_{ji}^{(t+1)}$, $\forall[j,i] \in \vec{E}$. By using the equality $R = I - J$, we can rewrite the expression as

$$\begin{aligned} z_{ji}^{(t+1)} &= \frac{R_{ij}^2}{1-R_{ij}^2}\left(h_i + \sum_{u\in N(i)\backslash j} z_{ui}^{(t)}\right) + \frac{R_{ij}}{1-R_{ij}^2}\left(h_j + \sum_{v\in N(j)\backslash i} z_{vj}^{(t)}\right) \\ &\overset{(a)}{=} \sum_{k=1}^{\infty} R_{ij}^{2k}\left(h_i + \sum_{u\in N(i)\backslash j} z_{ui}^{(t)}\right) + \sum_{k=1}^{\infty} R_{ij}^{2k-1}\left(h_j + \sum_{v\in N(j)\backslash i} z_{vj}^{(t)}\right), \end{aligned} \tag{3.4}$$

where in step a, we use the equality $\frac{1}{1-R_{ij}^2} = \sum_{k=0}^{\infty} R_{ij}^{2k}$.

Next we study the geometrical meaning of the parameters $\{z_{ji}^{(t)}, [j,i] \in \vec{E}, t \geq 1\}$. Using the fact that $z_{uk}^{(0)} = 0$ for all $[u,k] \in \vec{E}$, we have

$$z_{ji}^{(1)} = \sum_{k=1}^{\infty} R_{ji}^{2k} h_i + \sum_{k=1}^{\infty} R_{ji}^{2k-1} h_j, \quad [j,i] \in \vec{E}.$$

$z_{ji}^{(1)}$ can be taken as the the weight of all walks that travel only between node i and j and have the same end edge $[j,i]$. By induction, using equation 3.4, $z_{ji}^{(t)}$ can be taken as the weight of a subset of $\mathcal{W}_{ji}^{*}$ with the same end edge $[j,i]$. We denote the set of all walks contributed to the weight $z_{ji}^{(t)}$ as $\mathcal{W}_{ji}^{(t)}$. Each element $w \in \mathcal{W}_{ji}^{(t)}$ has $[j,i]$ as its end edge and length $|w| \geq 1$. For the special case that $t = 0$, we let $\mathcal{W}_{ji}^{(0)} = \emptyset$, where $\emptyset$ denotes the empty set. When $t \geq 1$, it can be easily shown that each element $w \in \mathcal{W}_{ji}^{(t)}$ is used only once in computing $z_{ji}^{(t)}$, that is, $z_{ji}^{(t)} = \gamma(\mathcal{W}_{ji}^{(t)})$. It should be noted that there exist elements of length greater than t (may be even infinite) in $\mathcal{W}_{ji}^{(t)}$. Based on the analysis, we obtain the following lemma about $z_{ji}^{(t)}$:

Lemma 1. *When J is walk-summable ($\rho(\bar{R}) < 1$) and $z_{ij}^{(0)} = 0$ for all $[i,j] \in \vec{E}$, the sequence $\{z_{ji}^{(t)} = \gamma(\mathcal{W}_{ji}^{(t)}) | t = 1, \ldots, \infty\}$ converges for any $[j,i] \in \vec{E}$. Further, the sequence $\{\bar{\gamma}(\mathcal{W}_{ji}^{(t)}) | t = 1, \ldots, \infty\}$ also converges for any $[j,i] \in \vec{E}$.*

Next we characterize the relation between $\mathcal{W}_{ji}^{(t)}$ and $\mathcal{W}_{ji}^{(t+1)}$ for any $[j,i] \in \vec{E}$. We present the result in lemma 2:

Lemma 2.

$$\mathcal{W}_{ji}^{(t)} \subseteq \mathcal{W}_{ji}^{(t+1)} \quad \forall[j,i] \in \vec{E}, \quad t = 0, 1, \ldots \tag{3.5}$$

Proof. Using the fact that $\mathcal{W}_{ji}^{(0)} = \emptyset$, we have

$$\mathcal{W}_{ji}^{(0)} \subseteq \mathcal{W}_{ji}^{(1)} \quad \forall [j, i] \in \vec{E}. \tag{3.6}$$

By induction using equations 3.4 and 3.6, it is immediate that equation 3.5 holds.

From equation 3.5, we know that the set $\mathcal{W}_{ji}^{(t)}$ grows along with t. This implies that the estimate $\hat{x}_i^{(t)}$ in equation 2.27 is increasingly accurate with t. The remaining task is to show if equation 3.3) holds for $t = \infty$. To achieve this goal, we have to show that for each $[j, i] \in \vec{E}$, the set $\mathcal{W}_{ji}^{(\infty)}$ is complete. That is, $\mathcal{W}_{ji}^{(\infty)}$ includes all the walks with end edge $[j, i]$. We define $\mathcal{V}_{ji}^{(t)}$ to be the set of all walks of length $t \geq 1$ with the common end edge (j, i). That is, for any $w \in \mathcal{V}_{ji}^{(t)}$, there are $|w| = t$ and $[w_{t-1}, w_t] = [j, i]$. We let $\mathcal{V}_i^{(t)} = \bigcup_{u \in N(i)} \mathcal{V}_{ui}^{(t)}$ where $t \geq 1$. The weight of the set $\mathcal{V}_i^{(t)}$ is equal to the quantity $[R^t h]_i$, that is, $\gamma(\mathcal{V}_i^{(t)}) = [R^t h]_i$, $t \geq 1$. We characterize the relation between $\mathcal{W}_{ji}^{(t)}$ and $\mathcal{V}_{ji}^{(t')}$ in lemma 3:

Lemma 3. *For any directed edge $[j, i] \in \vec{E}$, there is*

$$\mathcal{W}_{ji}^{(t)} \supseteq \bigcup_{k=1}^{t} \mathcal{V}_{ji}^{(k)} \quad t = 1, \ldots, \infty. \tag{3.7}$$

Proof. We use an induction argument to prove the lemma. When $t = 1$, $\mathcal{V}_{ji}^{(1)}$ has only one element $[j, i]$. It is straightforward from equation 3.4 that $\mathcal{W}_{ji}^{(1)} \supseteq \mathcal{V}_{ji}^{(1)}$. Next we assume that equation 3.7 holds for $t = m$. When $t = m + 1$, we have from equation 3.5 that $\mathcal{W}_{ji}^{(m+1)} \supseteq \bigcup_{k=1}^{m} \mathcal{V}_{ji}^{(k)}$. Thus, we need to show only that

$$\mathcal{W}_{ji}^{(m+1)} \supseteq \mathcal{V}_{ji}^{(m+1)}.$$

Suppose $w \in \mathcal{V}_{ji}^{(m+1)}$. Then one can always divide w into two segments $w = [w_1, w_2]$ such that w_2 involves only the node i and j and the end edge of w_1 (if there is one) is either $\{[u, i], u \in N(i) \backslash j\}$ or $\{[v, j], v \in N(j) \backslash i\}$. Using the fact that $|w_1| \leq t$ and equation 3.4, we have $w \in \mathcal{W}_{ji}^{(m+1)}$.

With these three lemmas, we are ready to present the main result regarding the convergence of the LiCD algorithm:

Theorem 1. *If the quadratic matrix J in equation 1.1 is walk-summable (i.e., $\rho(\bar{R}) < 1$) and $z_{ij}^{(0)} = 0$ for all $[i, j] \in \vec{E}$, then the iterates of the LiCD algorithm satisfy*

$$\|\hat{x}^{(t)} - x^*\|_2 \leq \frac{\rho(\bar{R})^{t+1}}{1-\rho(\bar{R})}\|\bar{h}\|_2, \tag{3.8}$$

where $\bar{h} = \left[|h_i|\right]$ and $\|\cdot\|_2$ denotes the spectral norm of a matrix (or, equivalently, matrix 2-norm).

Proof. Note that the computation of $\hat{x}_i^{(t)}$ involves the union of the sets $\bigcup_{u\in N(i)} \mathcal{W}_{ui}^{(t)} \bigcup\{i\}$. From lemma 3, we know that $\mathcal{W}_{ui}^{(t)} \supseteq \bigcup_{k=1}^{t} \mathcal{V}_{ui}^{(k)}$. Thus, the error $|\hat{x}_i^{(t)} - x_i^*|$ can be upper-bounded by

$$|\hat{x}_i^{(t)} - x_i^*| \leq \bar{\gamma}\left(\bigcup_{k=t+1}^{\infty} \mathcal{V}_i^{(k)}\right) = \left[\sum_{k=t+1}^{\infty} \bar{R}^k\bar{h}\right]_i.$$

Consequently, the error $\|\hat{x}^{(t)} - x^*\|_2$ can be upper-bounded by

$$\begin{aligned}\|\hat{x}^{(t)} - x^*\|_2 &\leq \left\|\left[\sum_{k=t+1}^{\infty} \bar{R}^k\bar{h}\right]\right\|_2 \\ &\leq \sum_{k=t+1}^{\infty} \|\bar{R}^k\|_2\|\bar{h}\|_2 \\ &\leq \frac{\rho(\bar{R})^{t+1}}{1-\rho(\bar{R})}\|\bar{h}\|_2.\end{aligned}$$

3.1.2 Initialization $\{z_{ij}^{(0)} \in \mathbb{R}\}$. We now consider the algorithm convergence for a general initialization. That is, $z_{ij}^{(0)} \in \mathbb{R}$ for any $[i, j] \in \vec{E}$. To achieve this goal, we use the result from lemma 1.

We first reformulate the updating expression 3.4 for any $[j, i] \in \vec{E}$ into vector form. Let $z^{(t)}$ denote the message (or parameter) vector at time t. The vector $z^{(t)}$ is of dimension $|\vec{E}|$, of which the components are indexed by the directed edges in $\vec{E}$. Correspondingly, equation 3.4 can be written in compact form as

$$z^{(t+1)} = Dy + Az^{(t)}, \tag{3.9}$$

where $y \in \mathbb{R}^{|\vec{E}|}$ is a vector with

$$y_{ji} = h_j, \quad [j,i] \in \vec{E}.$$

$D \in \mathbb{R}^{|\vec{E}|\times|\vec{E}|}$ is a matrix with

$$D_{\{ji,uk\}} = \begin{cases} \sum_{k=1}^{\infty} R_{ij}^{2k}, & \text{if } [j,i] \in \vec{E}, u = i, k = j \\ \sum_{k=1}^{\infty} R_{ij}^{2k-1}, & \text{if } [j,i] \in \vec{E}, u = j, k = i, \\ 0, & \text{otherwise} \end{cases}$$

and $A \in \mathbb{R}^{|\vec{E}|\times|\vec{E}|}$ is a matrix with

$$A_{\{ji,uk\}} = \begin{cases} \sum_{k=1}^{\infty} R_{ij}^{2k}, & \text{if } [u,k], [j,i] \in \vec{E}, k = i, j \neq u \\ \sum_{k=1}^{\infty} R_{ij}^{2k-1}, & \text{if } [u,k], [j,i] \in \vec{E}, k = j, u \neq i. \\ 0, & \text{otherwise} \end{cases}$$

Note that the spectral radius of A in equation 3.9 is essential for the convergence of the LiCD algorithm. When $z^{(0)} = 0$, it is immediate from lemma 1 that the infinite summation $\lim_{t\to\infty} z^{(t)} = \sum_{t=0}^{\infty} A^t D y$ is absolutely convergent. Therefore, the spectral radius of $\bar{A}$, where $\bar{A} = [|A_{ij}|]_{i,j=1}^{|\vec{E}|}$, must be less than 1, that is, $\rho(\bar{A}) < 1$. As a result, the spectral radius of A is also less than 1. We summarize the result in lemma 4:

Lemma 4. *If the quadratic matrix J in equation 1.1 is walk-summable (i.e., $\rho(\bar{R}) < 1$), then the matrix A has the properties $\rho(\bar{A}) < 1$ and $\rho(A) < 1$.*

For an arbitrary initialization $z^{(0)} \in \mathbb{R}^{|\vec{E}|}$, we have

$$z^{(t)} = \sum_{k=0}^{t-1} A^k D y + A^t z^{(0)}.$$

The property $\rho(A) < 1$ implies that the impact of the initialization $z^{(0)}$ on $z^{(t)}$ and further the estimate $\hat{x}^{(t)}$ decays exponentially over time. In other words, the two vectors $z^{(t)}$ and $\hat{x}^{(t)}$ become independent of the initialization

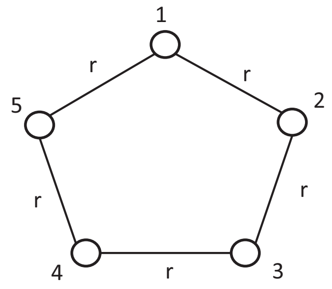

Figure 2: The five-cycle graph. The edge weights are as denoted by r in the graph.

$z^{(0)}$ as $t \to \infty$. Based on the above analysis, we present the convergence result in theorem 2:

Theorem 2. *If the quadratic matrix J in equation 1.1 is walk-summable (i.e., $\rho(\bar{R}) < 1$) and $z_{ij}^{(0)} \in \mathbb{R}$ for all $[i, j] \in \vec{E}$, then the LiCD algorithm converges to the optimal solution x^*, that is, $\hat{x}^{(t)} \to x^*$.*

Remark 1. The walk sumability of J is not a necessary condition for the convergence of the LiCD algorithm. Consider the following example where we construct a matrix J by using the five-cycle graph as shown in Figure 2. The matrix J has unit-diagonal and off-diagonal elements being the edge weights as indicated in the graph. We let $r = 0.6$ and $h_i = i$, $i \in V$. Correspondingly, $\rho(\bar{R}) = 1.2 > 1$. The matrix J is not walk-summable. The LiCD algorithm still converges for this particular example.

3.2 Asynchronous Convergence. The results of theorems 1 and 2 establish the algorithm convergence for the synchronous message-updating scenario. That is, all of the messages are updated at every time step in parallel. In this section, we study the convergence of the LiCD algorithm under an asynchronous model of message updating.

We consider a simple asynchronous model. That is, each time only one directed edge is randomly selected from $\vec{E}$. Then only the associated parameter along the directed edge is updated. At the same time, all the other messages remain the same. We make the assumption that any directed edge in the graph would be eventually selected if arbitrary delay is allowed in the algorithm.

In the following, we argue that theorems 1 and 2 can be extended to the asynchronous model. Our main idea is to show that any initialization $z^{(0)}$ has less and less impact on $z^{(t)}$ and $\hat{x}^{(t)}$ as the algorithm evolves. To achieve this goal, we let $y = 0$ in equation 3.9 (or, equivalently, $h = 0$) to remove the impact of h on $z^{(t)}$ and $\hat{x}^{(t)}$. We then show that $z^{(t)} \to 0$ as $t \to \infty$ for any initialization $z^{(0)}$.

Before formally presenting the argument, we introduce the weighted sup-norm used in Moallemi and Roy (2010) for distributed convex optimization. Suppose there is a positive vector $s \in \mathbb{R}^{|\vec{E}|}$ where each element corresponds to an edge in $\vec{E}$. For vectors $z \in \mathbb{R}^{|\vec{E}|}$, the weighted sup-norm is defined as

$$\|z\|_\infty^s = \max_{[i,j]\in\vec{E}} \frac{|z_{ij}|}{s_{ij}}. \tag{3.10}$$

For a matrix $A \in \mathbb{R}^{|\vec{E}|\times|\vec{E}|}$, the weighted sup-norm is given by

$$\|A\|_\infty^s = \max_{[j,i]\in\vec{E}} \frac{1}{s_{ji}} \sum_{[u,k]\in\vec{E}} s_{uk}|A_{\{ji,uk\}}|. \tag{3.11}$$

The selection of s is the key point in the following argument. We present the result regarding the vector s in lemma 5:

Lemma 5. *If the matrix J is walk-summable, then there exists a positive vector $s \in \mathbb{R}^{|\vec{E}|}$ such that*

$$\|\bar{A}\|_\infty^s < 1.$$

Proof. See the appendix for the proof.

Suppose at time t, a directed edge $[i,j] \in \vec{E}$ is selected. We consider using the latest information from neighbors of i and j to compute $z_{ij}^{(t)}$. Define $0 \le \tau_{uv}(t) < t$ to be the last time the directed edge ($[u,v] \in \vec{E}$, has been selected. Then from equation 3.9 and using the condition that $y = 0$, the parameter $z_{ij}^{(t)}$ is computed as

$$z_{ij}^{(t)} = \sum_{u\in N(i)\backslash j} A_{\{ij,ui\}} z_{ui}^{(\tau_{ui}(t))} + \sum_{v\in N(j)\backslash i} A_{\{ij,vj\}} z_{vj}^{(\tau_{vj}(t))}. \tag{3.12}$$

This expression can be written in vector form. By applying the weighted sup-norm $\|\cdot\|_\infty^s$ on the obtained vector form and using the property that $\|\bar{A}\|_\infty^s < 1$, we have

$$|z_{ij}^{(t)}|/s_{ij} < \max\left\{ \frac{\left|z_{ui}^{(\tau_{ui}(t))}\right|}{s_{ui}}, u \in N(i)\backslash j; \frac{\left|z_{vj}^{(\tau_{vj}(t))}\right|}{s_{vj}}, v \in N(j)\backslash i \right\}. \tag{3.13}$$

We use equation 3.13 to show that as $t \to \infty$, the vector $z^{(t)}$ approaches 0. Suppose at time t_0, the ratio $|z_{pq}^{(\tau_{pq}(t_0))}|/s_{pq}$ has maximum value:

$$\left|z_{pq}^{(\tau_{pq}(t_0))}\right|/s_{pq} = \max\left\{\left|z_{vu}^{(\tau_{vu}(t_0))}\right|/s_{vu}, [v,u] \in \vec{E}\right\} > 0.$$

For simplicity, we assume that only one element has the maximum value in the maximization operation. The analysis that follows can be easily extended to the general case. The assumption in the asynchronous model guarantees that there exists a time step $t_1 > t_0$ at which time the directed edge $[p, q]$ is selected. When equation 3.13 is applied, the maximum ratio at time t_1 is smaller than that at time t_0:

$$\max\left\{\left|z_{vu}^{(\tau_{vu}(t_1))}\right|/s_{vu}, [v,u] \in \vec{E}\right\} < \max\left\{\left|z_{vu}^{(\tau_{vu}(t_0))}\right|/s_{vu}, [v,u] \in \vec{E}\right\}.$$

Thus, as $t \to \infty$, the vector $z^{(t)}$ approaches 0 in the sense of the weighted sup-norm (i.e., $\|\cdot\|_\infty^s$).

Finally, since it is known that the estimate $\hat{x}^{(t)}$ is independent of any initialization $z^{(0)}$, the algorithm arrives at a stable solution. To further show that the stable solution is the optimal solution $x^* = J^{-1}h$, one may also use the walk-sum concept as in the synchronous message-passing scenario by letting $z^{(0)} = 0$. Since the procedure is similar to that in section 3.1, we omit the details here.

The asynchronous model allows only one directed edge to be activated at each time step. Thus, the information of (J, h) does not spread fast enough over the graph. To speed up the information flow, several directed edges can be activated at the same time for message updating as long as their associated nodes do not overlap. This procedure would require a scheduling plan to manage the activation of the directed edges.

We note that the asynchronous model considered here is more flexible than that of the BCD algorithms. In the implementation of a BCD algorithm for a discrete optimization problem, the parameters are updated block by block. On the other hand, the LiCD algorithm can update the parameters either block by block or one by one.

4 Fixing Convergence for General *J*

The work we have presented thus far establishes the convergence of the LiCD algorithm for a particular class of J. That is, the matrix J is walk-summable. However, in practice, J may be an arbitrary symmetric positive-definite matrix. Thus, it is of great interest to extend the LiCD algorithm with guaranteed convergence for a general matrix J.

In order to fix the convergence of the LiCD algorithm for a general matrix J, we follow the line of work proposed in Johnson et al. (2009). In particular, they propose a double-loop algorithm for the quadratic optimization problem, where the min-sum algorithm is taken as a subroutine. Their basic idea is to perform diagonal loading on J to obtain $J_\alpha = (1-\alpha)J + \alpha I$, $1 > \alpha \geq 0$. Then instead of solving $x^* = J^{-1}h$ directly, they considered a sequence of linear equations:

$$\hat{x}^{(t+1)} = J_\alpha^{-1}((1-\alpha)h + \alpha\hat{x}^{(t)}) \quad t = 0, 1, \ldots \tag{4.1}$$

Johnson et al. (2009) showed that by following the iteration in equation 4.1, $\hat{x}^{(t)}$ converges to $x^* = J^{-1}h$ for any initialization $\hat{x}^{(0)}$. Given $\hat{x}^{(t)}$ at time t, the new estimate $\hat{x}^{(t+1)}$ is computed by exploiting the min-sum algorithm. To guarantee the convergence of the min-sum algorithm, α is chosen to ensure that J_α is walk-summable. The algorithm involves two loops: the outer loop follows iteration 4.1, and the inner loop implements the min-sum algorithm for each time step in that equation.

In order to understand the double-loop algorithm better, we take a close look at equation 4.1. The iteration converges when the estimate $\hat{x}^{(t)}$ becomes stable: $\hat{x}^{(t)} = \hat{x}^{(t+1)} = \hat{x}^{(\infty)}$ for some t. In this situation, there is $\hat{x}^{(\infty)} = J_\alpha^{-1}((1-\alpha)h + \alpha\hat{x}^{(\infty)})$. Hence, $J\hat{x}^{(\infty)} = h$. To briefly summarize, equation 4.1 eventually reaches the optimal solution that minimizes the quadratic function, equation 1.1. The key point for the algorithm is to select the parameter α properly. Johnson et al. (2009) showed that as long as $\alpha > 1 - 1/\rho(\bar{R})$, J_α is walk-summable. For the special case that $\rho(\bar{R}) < 1$, $\alpha = 0$ corresponds to the min-sum algorithm without outer-loop iteration.

In this work, we exploit the LiCD algorithm as a subroutine in designing a double-loop algorithm. Similarly, we take equation 4.1 to be the outer-loop iteration. Since the LiCD and the min-sum algorithm share the same convergence condition, the LiCD-based double-loop algorithm would also converge to the optimal solution $x^* = J^{-1}h$. For a fixed parameter α in J_α, the convergence speed of the double-loop algorithm is determined by the inner-loop procedure. If the LiCD algorithm is efficient, the corresponding double-loop algorithm would also be efficient. Since the LiCD algorithm transmits only one parameter per message, the new double-loop algorithm also carries this property.

We note that instead of the LiCD algorithm, the Jacobi algorithm can also be used to design a Jacobi-based double-loop algorithm. The argument for the algorithm convergence is similar as above.

5 Experiments with Synthetic Data

We evaluated the proposed LiCD algorithm for both the synchronous and asynchronous message-updating scenarios. In the synchronous message-updating scenario, we implemented the Jacobi and the min-sum algorithms for performance comparison. In the experiments, we studied the impact of different graphical topologies on the performance of the three algorithms.

In the asynchronous scenario, we implemented the min-sum algorithm for performance comparison. The implementation is similar to that of the LiCD algorithm (see section 3.2). We note that it may not be appropriate to take the Gauss-Seidel algorithm as a reference in this scenario. This is because the LiCD and Gauss-Seidel algorithms transmit parameters by following different schemes. For the Gauss-Seidel algorithm, once a node i updates its own estimate $\check{x}_i$ of x_i^*, it broadcasts the new value to all its neighbors in the graph. On the other hand, the parameters of the LiCD algorithm are edge specific. Once a directed edge $[i, j] \in \vec{E}$ is activated, the parameter $\beta_{i\setminus j}$ is computed at node i and transmitted to node j for updating z_{ij} (see section 2.2).

5.1 Synchronous Message Updating. In the experiments, we investigated three types of graphical topologies: graphs with a line structure, graphs with one cycle, and general graphs with more cycles. Our main purpose is to study how the performance of the LiCD is affected by different graphical topologies.

5.1.1 Graph with a Line Structure:. In the first experiment, we considered the simplest graph, that is, the graph with a line structure. We set the number of nodes to be 10: $|V| = 10$. The off-diagonal components in J and those in h were generated from the gaussian distribution $N(0, 0.25)$. The diagonal elements of J were set to 1. Only J matrices that were positive definite were selected for test. From Johnson et al. (2009), we know that for a graph with a tree structure, the associated J matrix is walk-summable. Thus, the Jacobi, LiCD, and min-sum algorithms are guaranteed to converge to the optimal solution. To terminate the iterations of the three algorithms, the infinite norm between an estimate and the optimal solution x^* was measured. The convergence threshold were set as 10^{-10}.

Seven pairs of (J, h) were tested: the experimental results are displayed in Table 2. The min-sum algorithm is most efficient in terms of convergence speed. Further, the number of iterations for the min-sum algorithm is always equal to 9 (the diameter of the graph), which coincides with the results of Pearl (1988). For each case, the LiCD algorithm took a few more iterations than the min-sum algorithm. On the other hand, the LiCD algorithm outperforms the Jacobi algorithm significantly.

5.1.2 Graph with One Cycle. In the second experiment, we considered graphs with one cycle. The elements of J and h were generated as in the first experiment. Since there is a cycle in the graph, the generated matrix J might not be walk-summable. We applied the double-loop scheme proposed in Johnson et al. (2009) to fix the convergence of the three algorithms. As we

Table 2: Number of Iterations of the Three Algorithms for Seven Realizations of (J, h) with $|V| = 10$.

	1	2	3	4	5	6	7
Jacobi	115	125	306	568	397	64	95
LiCD	38	40	74	81	143	19	34
Min-sum	9	9	9	9	9	9	9

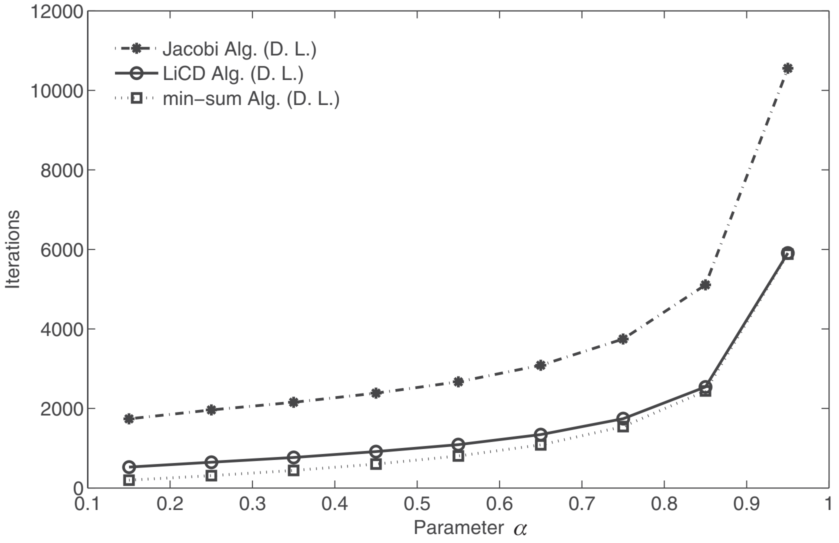

Figure 3: Impact of the parameter α on the performance of the three algorithms for a graph of 10 nodes with one-cycle structure. D. L.: double-loop.

explained in section 4, the outer loop follows the iteration in equation 4.1, and the inner loop implements each one of the three algorithms for each time step in equation 4.1. The parameter α in this equation was selected from the range $(1 - 1/\|\bar{R}\|_\infty, 1)$. Again to terminate the iterations of the algorithms, we measured the infinite norm between an estimate and the optimal solution. Convergence thresholds for the outer and inner loops were set as 10^{-5} and 10^{-10}, respectively.

The results of the three algorithms for a particular pair (J, h) are displayed in Figure 3. The number of nodes (or, equivalently, the dimension of h) was set as $|V| = 10$. It is seen that the LiCD and min-sum algorithms converge much faster than the Jacobi algorithm for different α values. Also, the performance of the LiCD and min-sum algorithms is getting close as α increases.

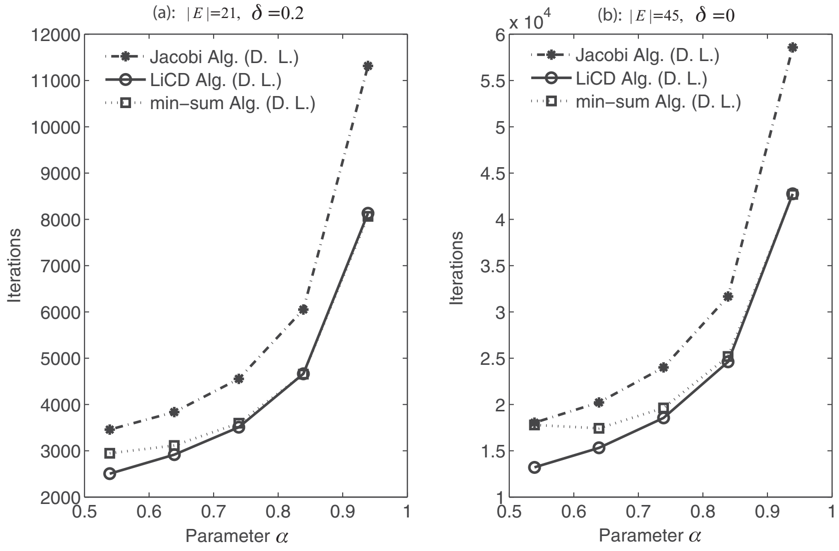

Figure 4: Impact of the parameter α on the performance of the three algorithms for a dynamic graph with a fixed number ($|V| = 10$) of nodes and increasing edges. D. L.: double-loop.

5.1.3 General Graph with More Cycles. In the third experiment, we considered general graphs with more cycles. We note that it was not easy to generate a (random) sparse J matrix that is positive definite. In order to obtain such a J matrix, we first generated a random matrix B, of which the components were realized from the gaussian distribution $N(0, 1)$. After that, a positive-definite matrix C with unit diagonal was computed by normalizing the matrix product $B^\top B$. Finally, we constructed a sparse matrix J by replacing the components of C that were within a range $[-\delta, \delta]$ with zeros, where δ was a threshold. The threshold δ determines the sparsity of the matrix J. If $\delta = 0$, we have $J = C$, corresponding to a fully connected graph. The convergence thresholds were the same as in the second experiment.

In the evaluation of the three algorithms, we chose two different values for δ for a particular pair (C, h), that is, $\delta = 0.2, 0$. The number (or the dimension of h) of nodes was $|V| = 10$. For the first case, that $\delta = 0.2$, $(|V|, |E|) = (10, 21)$ corresponds to a sparse graph with multicycles. For the second case that $\delta = 0$, $(|V|, |E|) = (10, 45)$ corresponds to a fully connected graph. The experimental results are shown in Figure 4. It is seen that the LiCD algorithm outperforms the Jacobi and min-sum algorithms for different α parameters in both cases.

To briefly summarize the three experiments, the efficiency of the LiCD algorithm is related to the graphic topologies. As a graph becomes dense

Table 3: Number of Iterations of the Two Algorithms for Seven Pairs of (J, h).

	1	2	3	4	5	6	7
$((\lvert V\rvert, \lvert E\rvert))$	(10, 21)	(10,23)	(15,44)	(15,48)	(20,61)	(20,130)	(30,327)
LiCD	630	684	1590	1694	1771	4620	11,260
min-sum	697	632	1590	1893	1804	4435	13,012

(the number of edges increases), the performance of the LiCD algorithm improves compared to that of the min-sum algorithm. For a graph with multiple cycles, the LiCD algorithm may outperform the Jacobi and min-sum algorithms with regard to the number of iterations. By taking into account the fact that the LiCD algorithm transmits only one parameter per message, the LiCD algorithm may be a good approach for graphs with multiple-cycles.

5.2 Asynchronous Message Updating. In the experiment, we applied a simple asynchronous message-updating scheme. That is, each time we activate only one node. Suppose node i is active at a time step. Node i first selects a node from its neighbors uniformly. The probability of selecting each neighboring node is thus $\frac{1}{d_i}$. Suppose node j is selected from $N(i)$. For the LiCD algorithm, node i computes and transmits the parameter $\beta_{i\backslash j}$ to node j for updating the parameter z_{ij}. For the min-sum algorithm, node i computes and transmits μ_{ij} and $\tilde{z}_{ij}$ to node j. In the next iteration, node j is taken to be active. This scheme in fact specifies a random walk on the graph. If time allows, all the directed edges in $\vec{E}$ eventually would be visited, which guarantees the convergence of the LiCD and min-sum algorithms. Since the Gauss-Seidel algorithm does not fit into this transmission scheme, we did not evaluate the algorithm.

The generation of (J, h) was similar to the case of a general graph with more cycles in the synchronous message-updating scenario. In the experiment, we chose only the J matrices that are walk-summable for evaluation. The convergence threshold was set as 10^{-5}.

Seven pairs of (J, h) were tested; the experimental results are displayed in Table 3. We emphasize that each of the corresponding seven graphs has multiple cycles. It is seen that both the LiCD and min-sum algorithms have comparable convergence speeds. This convergence property of the LiCD algorithm is the same as the case of a general graph with more cycles in the parallel message-updating scenario.

Remark 2. In the asynchronous message-updating experiment, we also tested the LiCD algorithm for J matrices that were not walk-summable without using the double-loop scheme. Unfortunately, we found that the algorithm may fail for some J matrices. This property is different from that of the Gauss-Seidel algorithm, which converges for any positive-definite matrix J.

6 Application to Distributed Speech Enhancement in Wireless Microphone Networks

6.1 Problem Formulation. Consider a wireless microphone network (WMN) with n microphones, whose acoustic signals are windowed and transformed to the spectral domain using a discrete Fourier transform (DFT). We assume the presence of a single target source degraded by acoustical additive noise uncorrelated with the source. Let $Y = [Y_1, \ldots, Y_n]^\top$, denote a vector containing the stacked noisy DFT coefficients for each of the n microphones for a particular time frame and frequency bin.[3] The vector Y can be expressed as (Brandstein & Ward, 2001),

$$Y = Sd + N,$$

where (S, d, N) denote the target speech DFT coefficient, the (frequency-dependent) propagation vector, and the vector containing noise DFT coefficients, respectively. In this work, we assume that d is given. In addition, we assume that a global timing is available (e.g., by broadcast).

In order to estimate the target DFT coefficient S (for each frequency bin within each time frame), one can apply a spatial filter w to the noisy DFT coefficients, leading to an estimate of the clean speech signal $\hat{S} = w^*Y$, where $(\cdot)^*$ indicates Hermitian transposition. One particular choice for the spatial filter w leads to the so-called MVDR beam former, of which the coefficients are given by Brandstein and Ward (2001),

$$w_{MVDR} = \frac{R_N^{-1} d}{d^* R_N^{-1} d}, \tag{6.1}$$

where R_N is the autocorrelation matrix of the noise vector N, of which the diagonal elements are denoted as $(\sigma^2_{N_1}, \ldots, \sigma^2_{N_n})$. We note that in the computation of w_{MVDR}, a matrix inversion is required, which leads to the application of message-passing algorithms.

We consider applying the LiCD algorithm to compute the quantity $R_N^{-1}d$ in equation 6.1. It is worth noting that both R_N and d in that equation are defined in the complex domain while (J, h) in equation 1.1 are defined in the real domain. By following a similar argument in section 3, one can easily extend theorems 1 and 2 to the complex domain. That is, if R_N in equation 6.1

[3]In this letter, we assume that DFT coefficients are independent across time and frequency and therefore neglect time and frequency indices for ease of notation.

is walk-summable, the LiCD algorithm converges to the quantity $R_N^{-1}d$. If R_N is not walk-summable, one can apply the LiCD-based double-loop algorithm to compute $R_N^{-1}d$.

To save transmission power for microphones, we relax the autocorrelation matrix R_N so that the resulting matrix R'_N is walk-summable, thus avoiding the double-loop scheme. In this way, the convergence speed of the LiCD algorithm is significantly improved at the cost of degraded performance compared to that of the MVDR beamformer. In particular, we construct R'_N as $R'_N = (1-\eta)R_N + \eta\mathrm{diag}(\sigma^2_{N_1}, \ldots, \sigma^2_{N_n})$, where the parameter η is chosen to make R'_N walk-summable. Correspondingly, the spatial filter w_η is given by

$$w_\eta = \frac{R'^{-1}_N d}{d^* R'^{-1}_N d}. \tag{6.2}$$

We note that for the speech enhancement in wireless sensor networks, a speech signal is generally processed frame by frame. Each frame consists of a set of frequency bins, leading to a set of matrix inversions (one for each frequency bin). The LiCD algorithm can process the set of matrix inversions for each frame in parallel. As a result, for each iteration, a microphone can transmit a set of parameters (one for each frequency bin) to one of its neighbors at once, improving the transmission efficiency (i.e., in practice, extra data are needed for establishing a transmission connection).

6.2 Experimental Results. In this section we discuss experimental results obtained by computer simulations. The microphone network consisted of nine microphones lying on a 2D rectangular grid, as depicted in Figure 5a. One of the nine microphones is denoted by a for reference. The distances between neighboring microphones were set to 2 meters. The nine microphones formed a fully connected graph for distributed speech enhancement. We considered the situation that there were one speaker and three noise sources within the microphone network. Their locations were generated randomly, as illustrated in Figure 5a.

The parameters in the experiment were set as follows. The sampling frequency was $f_s = 16$ kHz. Each frame contained 400 samples, corresponding to a speech segment of 25 ms. A 50% overlapped Hanning window was used. The three noise sources were simulated by independent white gaussian noise. The noise correlation matrices R_N for different frequency bins were estimated beforehand. A speech signal of 20 s was processed by the Jacobi, LiCD, and min-sum algorithms. The convergence threshold for the three algorithms was set as 10^{-5}. As described in section 6.1, the parameter η for each frequency bin was chosen such that R'_N was walk-summable.

The SNR for microphone a in the network before speech enhancement is −8.2 dB. The SNRs for all the microphones after speech enhancement

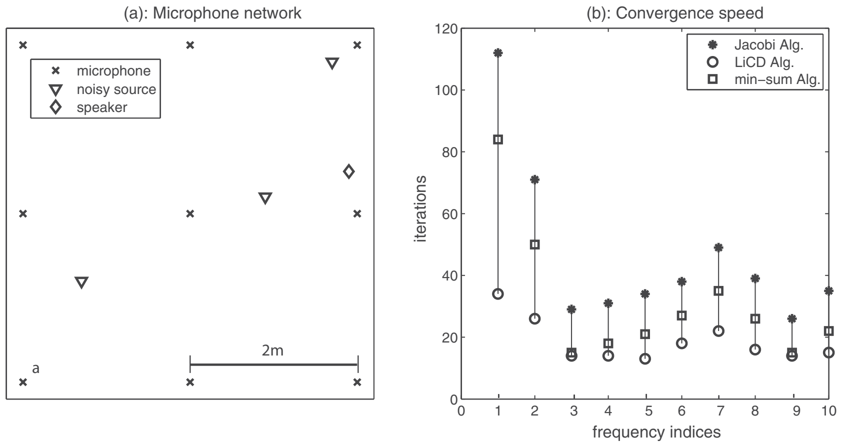

Figure 5: (a) A microphone network with nine microphones, one speaker, and three noisy sources. One of the nine microphones is denoted by *a* for reference. (b) The convergence speeds of the three algorithms for the first 10 frequency bins. In particular, frequency 1 corresponds to the DC component.

are the same, which is 12.6 dB. Figure 5b demonstrates the (time) average numbers of iterations needed for the three algorithms in each of the first 10 frequency bins. It is seen that the LiCD algorithm outperforms the Jacobi and min-sum algorithms. The Jacobi algorithm performs the worst in each frequency bin.

7 Conclusion

We have proposed a new message-passing algorithm for quadratic optimization by performing linear coordinate-descent operations. Compared to the min-sum algorithm of which the messages are quadratic functions, the LiCD algorithm has linear-functional messages like the Jacobi algorithm. We have shown that the LiCD has the same convergence condition as the min-sum algorithm. Similar to the min-sum algorithm, the LiCD algorithm works by performing either synchronous or asynchronous message passing.

Experimental results show that for arbitrary graphs, the LiCD algorithm outperforms the Jacobi algorithm significantly with regard to convergence speed, even though the two algorithms transmit the same number of parameters per iteration. For graphs with multiple cycles, the LiCD algorithm has comparable convergence speed to the min-sum algorithm, but at lower complexity in terms of computation and number of transmission parameters.

We note that in the literature, the Jacobi algorithm was extended to the Jacobi-relaxation algorithm to cover all positive-definite matrices J. In further work, we will consider extending the LiCD algorithm in a similar way as the Jacobi-relaxation algorithm to replace the LiCD-based double-loop algorithm.

Appendix: Proof of Lemma 5

We assume, without loss of generality, that the graph $G = (V, E)$ is connected (otherwise we can treat each connected subgraph separately). In this case, the matrix $\bar{R}$ is irreducible and nonnegative.

Next, we study the properties of the matrix $\bar{A}$. If every node in the graph has at least two neighbors (i.e., $N(i) \geq 2$, $\forall i \in V$), then the matrix $\bar{A}$ is irreducible and nonnegative. In this situation, one can directly apply the Perron-Frobenius theorem (Horn & Johnson, 1990) to $\bar{A}$. In particular, there exists a positive vector $s \in \mathbb{R}^{|\vec{E}|}$ and a scalar $\lambda = \rho(\bar{A})$, so that

$$\bar{A}s = \lambda s.$$

By using $0 < \lambda < 1$, we have $\|A\|_{\infty}^{s} < 1$.

For the case that the graph G is more general (i.e., some nodes may have only one neighbor), the matrix $\bar{A}$ is reducible. To see this, we suppose that a node $i \in V$ has only one neighbor j (i.e., $J_{ij} \neq 0$). It is immediate from the updating expressions that the message z_{ji} has no contribution to all other messages $\{z_{uv}, [u, v] \neq [j, i], [u, v] \in \vec{E}\}$. Correspondingly, the column of the matrix $\bar{A}$ indexed by $[j, i]$ is zero. Thus, by proper permutation on the columns of $\bar{A}$, the matrix takes the form

$$\bar{A}_{per} = \begin{bmatrix} \mathbf{0}_{\mathbf{k}\times\mathbf{k}} & \bar{A}_1 \\ \mathbf{0}_{|\vec{E}-k|\times k} & \bar{A}_2 \end{bmatrix}.$$

By identifying all the zero columns in $\bar{A}$, the submatrix $\bar{A}_2$ is irreducible and nonnegative. Further, the spectral radius of $\bar{A}_2$ is less than 1 (i.e., $\rho(\bar{A}_2) < 1$). Again from the Perron-Frobenius theorem, there exists a positive vector $s_2 \in \mathbb{R}^{|\vec{E}|-k}$ and a scalar $\lambda_2 = \rho(\bar{A}_2)$, so that

$$\bar{A}_2 s_2 = \lambda_2 s_2.$$

With s_2 at hand, one can easily find a positive vector $s_1 \in \mathbb{R}^k$ such that for $s = [s_1^\top \; s_2^\top]^\top$, $\|A\|_{\infty}^{s} < 1$. This completes the proof.

References

Bertrand, A., & Moonen, M. (2012). Distributed node-specific LCMV beamforming in wireless sensor networks. *IEEE Transactions on Signal Processing, 60*, 1, 233–246.

Bertsekas, D. P., & Tsitsiklis, J. N. (1997). *Parallel and distributed computation: Numerical methods*. Belmont, MA: Athena Scientific.

Boyd, S., Ghosh, A., Prabhakar, B., & Shah, D. (2006). Randomized gossip algorithms. *IEEE Trans. Inform. Theory*, *52*, 2508–2530.

Brandstein, M., & Ward, D. (2001). *Microphone arrays*. New York: Springer.

Globerson, A., & Jaakkola, T. (2008). Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In J. C. Platt, D. Köller, Y. Singer, & S. Roweis (Eds.), *Advances in neural information processing systems*, *21* (pp. 553–560). Cambridge, MA: MIT Press.

Heusdens, R., Zhang, G., Hendriks, R. C., Zeng, Y., & Kleijn, W. B. (forthcoming). Distributed MVDR beamforming for (wireless) microphone networks using message passing. In *Proceedings of the International Workshop on Acoustic Signal Enhancement*.

Horn, R. A., & Johnson, C. R. (1990). *Matrix analysis*. Cambridge: Cambridge University Press.

Johnson, J. K., Bickson, D., & Dolev, D. (2009). Fixing convergence of gaussian belief propagation. In *Proceedings of the 2009 IEEE International Symposium on Information Theory* (pp. 1674–1678). Piscataway, NJ: IEEE.

Johnson, J. K., Malioutov, D. M., & Willsky, A. S. (2006). Walk-sum interpretation and analysis of gaussian belief propagation. In Y. Weiss, B. Schölkopf, & J. Platt (Eds.), *Advances in neural information processing systems*, *18*. Cambridge, MA: MIT Press.

Lauritzen, S. L. (1996). *Graphical models*. New York: Oxford University Press.

Malioutov, D. M., Johnson, J. K., & Willsky, A. S. (2006). Walk-sums and belief propagation in gaussian graphical models. *J. Mach. Learn. Res.*, *7*, 2031–2064.

Moallemi, C. C., & Roy, B. V. (2009). Convergence of min-sum message passing for quadratic optimization. *IEEE Trans. Inf. Theory*, *55*, 2413–2423.

Moallemi, C. C., & Roy, B. V. (2010). Convergence of min-sum message passing for convex optimization. *IEEE Trans. Inf. Theory*, *56*, 2041–2050.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Francisco: Morgan Kaufmann.

Rusmevichientong, P., & Roy, B. B. (2001). An analysis of belief propagation on the turbo decoding graph with gaussian densities. *IEEE Trans. Inf. Theory*, *47*, 745–765.

Sontag, D., Globerson, A., & Jaakkola, T. (2011). Introduction to dual decomposition for inference. In S. Sra, S. Nowozin, & S. J. Wright (Eds.), *Optimization for Machine Learning*. Cambridge, MA: MIT Press.

Weiss, Y., & Freeman, W. T. (2001). Correctness of belief propagation in gaussian graphical models of arbitrary topology. *Neural Computation*, *13*, 2173–2200.

Wolfe, M. A. (1978). *Numerical methods for unconstrained optimization*. New York: Van Nostrand Reinhold.

Received November 23, 2011; accepted June 21, 2012.