DCG: Distributed Conjugate Gradient for Efficient Linear Equations Solving

Haodi Ping, Yongcai Wang, and Deying Li

Abstract—Distributed algorithms to solve linear equations in multi-agent networks have attracted great research attention and many iteration-based distributed algorithms have been developed. The convergence speed is a key factor to be considered for distributed algorithms, and it is shown dependent on the spectral radius of the iteration matrix. However, the iteration matrix is determined by the network structure and is hardly pre-tuned, making the iterative-based distributed algorithms may converge very slowly when the spectral radius is close to 1. In contrast, in centralized optimization, the Conjugate Gradient (CG) is a widely adopted idea to speed up the convergence of the centralized solvers, which can guarantee convergence in fixed steps. In this paper, we propose a general distributed implementation of CG, called DCG. DCG only needs local communication and local computation, while inheriting the characteristic of fast convergence. DCG guarantees to converge in $4Hn$ rounds, where $H$ is the maximum hop number of the network and $n$ is the number of nodes. We present the applications of DCG in solving the least square problem and network localization problem. The results show the convergence speed of DCG is three orders of magnitude faster than the widely used Richardson iteration method.

Index Terms—distributed algorithm, conjugate gradient, linear equations, network localization, least square problem

I. INTRODUCTION

In many multi-agent applications, the underlying problem can be reduced to solving a system of linear equations [1]. Because the autonomous networked agents are usually discretely deployed, each agent only has the access to communicate with direct neighbors. Moreover, in certain scenarios, each agent only desires its own state. Such characteristics consequently give rise to distributed solvers for linear systems. Different from the centralized solvers, distributed solvers usually adopt iterative manners. The key feature of iterative approaches is linear iterations, where each agent receives states of direct neighbors; updates and sends its own state. The barycentric linear localization algorithm [2], is a typical iteration-based method.

The convergence speed is a crucial factor of the iteration-based distributed algorithms, which determines whether the distributed algorithm can be used when the application requires a fast response. However, the convergence rate of many iterative methods, e.g., Jacobi iteration, Gauss-Seidel iteration, and Richardson iteration [3], are characterized by the spectral radius of the iteration matrix. However, the spectral radius is determined by the network topology and is difficult to be pre-tuned. Thus the convergence speed is highly uncertain and may be very slow in some network states.

To guarantee fast convergence of distributed algorithms in solving linear equations, this study explores the idea of Conjugate Gradient (CG) [4]. CG has two desired properties for solving the linear systems: 1) CG converges to the exact solution after a finite number of iterations, which is not larger than the size of the system matrix; 2) CG is suited for solving linear systems with large and sparse system matrices. But CG is essentially a centralized solver. In pursuit of distributed implementation, we design an efficient protocol to synchronize the necessary vectors to update the residual. Our distributed CG, i.e., DCG remarkably speeds up convergence by paying limited neighborhood communication costs.

Fig. 1. The convergence trails in $\mathbb{R}^2$. The marker ‘+’ represents the start a trail. The marker ‘*’ and ‘◦’ represent the converged estimation $\hat{x}_i$ and the ground truth $x_i$, respectively.

Fig. 2. The mean square error w.r.t. iteration rounds.

The rest of the paper is organized as follows. We formulate the problem and present related algorithms in Section II. DCG is proposed in Section III. Applications of DCG are discussed in Section IV. DCG is evaluated in Section V. The paper is concluded with further discussions in Section VI.
Notations: Throughout the paper, let $AX = b$ denote a system of linear equations, where $A \in \mathbb{R}^{n \times n}$ is the coefficient matrix, $b \in \mathbb{R}^{n \times d}$ is the right-hand side vector, and $X \in \mathbb{R}^{n \times d}$ is the vector of unknowns. $n$ is the number of variables and $d$ is the spatial dimension of a vector, $d \in \{2, 3\}$. The vector $A_i = [A_{i1} \cdots A_{in}]$ denotes the $i$th row of $A$, $i \in \{1 \cdots n\}$.

II. Preliminaries

A. Problem Formulation

Consider a network of $n$ agents $V = \{v_1, \cdots, v_n\}$, each agent $v_i$ is capable of communicating with the agents within its reception range $R$. Let $E$ be the set of edges and $(i, j) \in E$ if the distance between $v_i$ and $v_j$ is not larger than $R$. Then the multi-agent network can be represented as a graph $G = (V, E)$. The neighbors of $v_i$ is denoted by $N_i$, $j \in N_i$ if $(i, j) \in E$.

Problem: Assume that $A$ is non-singular. Let $X^*$ denote the unique solution satisfying $AX = b$. Suppose each agent $v_i$ holds a state vector $\hat{x}_i \in \mathbb{R}^d$. Initially, $v_i$ knows $A_{i:}$ and $b_{i:}$. The problem is to devise a local rule for each agent to update its state $\hat{x}_i$ leveraging the local communication with agents in $N_i$ so that $\hat{x}_i(t)$ converges to $x^*_i$ within finite $t$.

B. Related Work

The basic idea of the iterative methods is as follows. Given an initialization $\hat{x}(0)$, generate an iteration sequence $\{\hat{x}(t)\}_{t=0}^\infty$ in a certain manner, so that:

$$\lim_{t \to \infty} \hat{x}(t) = x^* \triangleq A^{-1}b. \quad (1)$$

Generally, the state update of an iterative method can be represented as:

$$\hat{x}(t + 1) = \phi_k(\hat{x}(t), \hat{x}(t - 1) \cdots, \hat{x}(0), A, b), \quad (2)$$

where $\hat{x}(0) = \phi_0(A, b)$ or $\hat{x}(0)$ is selected manually, $\phi_k$ is called the iteration function. The specific design of iterative functions are based on the matrix splitting.

Definition 1 (Matrix Splitting). Suppose a non-singular matrix $A \in \mathbb{R}^{n \times n}$, a split of matrix $A$ is defined as $A = M - N$, where $M$ is also non-singular.

Consider a general linear system:

$$AX = b, \quad (3)$$

where $A$ is non-singular. $A$ can be transformed as follows through matrix splitting $A = M - N$.

$$MX = NX + b. \quad (4)$$

Then, we can construct an iteration function as:

$$M\hat{x}(t + 1) = N\hat{x}(t) + b, \quad (5)$$

which is equivalent to:

$$\hat{x}(t + 1) = M^{-1}N\hat{x}(t) + M^{-1}b \triangleq G\hat{x}(t) + g. \quad (6)$$

$G = M^{-1}b$ is called the iteration matrix. It is straightforward that different iterative methods can be constructed by varying $M$.

In Jacobi iteration, $A$ is split as:

$$A = D - L - U. \quad (7)$$

$D$ is the diagonal component of $A$. $-L$ and $-U$ are the upper triangle component and the lower triangle component of $A$, respectively. Then, the Jacobi iteration is specified as:

$$\hat{x}(t + 1) = D^{-1}(L + U)\hat{x}(t) + D^{-1}b. \quad (8)$$

From the local behavior of an individual agent $v_i$, the state update is:

$$\hat{x}_i(t + 1) = \frac{1}{A_{ii}} \left( b_i - \sum_{j=1, j \neq i}^n A_{ij}\hat{x}_j(t) \right) \quad (9)$$

In Gauss-Seidel iteration, the iteration function is designed as:

$$\hat{x}(t + 1) = (D - L)^{-1}U\hat{x}(t) + (D - L)^{-1}b. \quad (10)$$

The local update of agent $v_i$’s state is:

$$\hat{x}_i(t + 1) = \frac{1}{A_{ii}} \left( b_i - \sum_{j=1}^{i-1} A_{ij}\hat{x}_j(t) - \sum_{j=i+1}^n A_{ij}\hat{x}_j(t) \right) \quad (11)$$

Another brief iteration method is the Richardson iteration when $A$ is symmetric positive definite. The iteration function is:

$$\hat{x}(t + 1) = \hat{x}(t) + \omega(b - A\hat{x}(t)). \quad (12)$$

The state of agent $v_i$ is updated as:

$$\hat{x}_i(t + 1) = \hat{x}_i(t) + \omega \left( b_i - \sum_{j=1}^n A_{ij}\hat{x}_j(t) \right) \quad (13)$$

where $\omega$ is a non-negative scalar and is suggested to be $\frac{2}{\lambda_{\max} + \lambda_{\min}}$. Similar methods include the Successive Over-Relaxation (SOR) iteration, the Symmetric SOR (SSOR) iteration, the Accelerated OR (AOR) iteration, the Symmetric AOR (SAOR) etc $[5]$.

C. Convergence and Convergence Rate

Since the aforementioned methods are iterative, a crucial issue is to guarantee iteration convergence. For a general iteration function as in $[6]$, its convergence is guaranteed by the following theorem.

Theorem 1. The iterates formulated by $\hat{x}(t + 1) = G\hat{x}(t) + b$ converges for any $\hat{x}(0)$, if and only if $\rho(G) < 1[6]$. $\rho(G)$ is the spectral radius of the iteration matrix $G$. See Theorem 4.1 of $[6]$ for the proof.

Apart from knowing when the iteration converges, it is also desirable to explore how fast it converges. Saad $[6]$ presented that the convergence rate $\tau$ is the natural logarithm of the inverse of the spectral radius.
\[ \tau = \ln \frac{1}{\rho(G)} = -\ln \rho(G). \]  
(14)

It can be seen that these iterative methods converge slowly when \( G \) has unstable eigenvalues.

III. DCG: A GENERAL DISTRIBUTED CONJUGATE GRADIENT IMPLEMENTATION

The most attractive feature of CG is that it converges within a fixed round of iterations. But CG is essentially a centralized gradient-based solver for linear equations. See Section 6 of [6] for the original CG algorithm. Although parallel or distributed CG algorithms have been reported for allocating computational loads in cloud computing and estimating spectrum in sensor networks[8], the general distributed CG has not been well touched.

The key difficulty in implementing Distributed CG (DCG) is that the state \( \hat{X}_i \) is updated based on several vectors, however, \( v_i \) only knows the \( i \)th element of these vectors. Specifically, \( \hat{X}_i(t) = \hat{X}_i(t-1) + \alpha_i(t)d_i(t) \). \( \alpha_i(t) \) and \( d_i(t) \) are calculated using the direction vector \( d(t) \) and the residual vector \( r(t) \). But \( v_i \) only knows \( d_i \) and \( r_i \).

A. Vector Synchronization

To supplement the necessary information, we design the \textit{Synchronize\_Vector}(\( z_i \)) protocol, through which \( v_i \) can gather any complete vector \( z \) by constantly exchanging respective elements with neighbors (Line 21-31 of Algorithm 1). \( H \) is the largest number of hops throughout the network. So the synchronization can be finished in \( H \) rounds. \( H \) can be set to \( n \) if it is not given at network deployment. The operation ‘merge’ (Line 29) means selecting all non-zero elements of the input vectors and stacking them as a new vector while maintaining their original indexes.

B. Distributed Conjugate Gradient

DCG is detailed as \textbf{Function DCG} (Line 2-18 of Algorithm 1). At initialization (Line 2), the state \( \hat{X}_i(0) \) is set to 0. The direction vector component \( d_i \) and the residual vector component \( r_i \) are set to 0 and \( -b_i \), respectively. At each iteration \( t \), the local behavior of a node \( v_i \) is as follows:

- **Update Residual** (Line 9). The residual \( r_i \) is updated by \( r_i(t) = (A\hat{X}_i(t-1) - z_i) \), which is realized by \( r_i(t) = -b_i + \sum_{j\in\mathcal{N}_i} A_{ij}\hat{X}_j(t-1) \). \( A_{i,j} \) and \( b_i \) are known to \( v_i \) at the beginning. \( \hat{X}_j(t-1) \) is the state of neighbor \( v_j \) obtained by local communication.

- **Check Residual** (Line 8). CG theoretically completes after \( n \) iterations [3]. However, due to the accumulated floating point rounding off errors, the residual and the direction gradually lose accuracy. Thus, DCG terminates by checking whether \( r(t)^T r(t) < \varepsilon \). The threshold \( \varepsilon \) is an empirical value based on accuracy requirement. \( r(t) \) is obtained by: \( r(t) = \text{Synchronize\_Vector}(r_i(t)) \).

After knowing the synchronized \( r \), the squared residual is calculated as \( r(t)^T r(t) = \sum_{i=1}^{n} r_i(t)^2 \).

- **Update Direction** (Line 12). \( d_i \) needs another vector \( r(t-1) \) to be updated. It is obtained by \textit{Synchronize\_Vector}(\( r_i(t-1) \)). Then, \( r(t-1)^T r(t-1) = \sum_{i=1}^{n} r_i(t-1)^2 \) and \( d_i \) can be calculated locally.

- **Update Step Size** (Line 14). Represent the numerator and denominator of \( \alpha_i \) as \textit{numerator} = \( d(t)^T r(t) \) and \textit{denominator} = \( d(t)^T (A\hat{X}(t)) \), respectively. The direction vector \( d(t) \) is obtained by \textit{Synchronize\_Vector}(\( d_i(t) \)). So the numerator can be known as \textit{numerator} = \( \sum_{i=1}^{n} d_i(t) r_i(t) \). An intermediate vector \( T(t) = [T_1(t) \cdots T_n(t)]^T \) is introduced to calculate \textit{denominator}. By communicating with neighbors, \( v_i \) calculates \( T_i(t) = \sum_{j\in\mathcal{N}_i} A_{ij} d_j(t) \). The complete \( T(t) \) is obtained by \textit{Synchronize\_Vector}(\( T_i(t) \)). So \textit{denominator} = \( \sum_{i=1}^{n} d_i(t) T_i(t) \). Then \( \alpha_i = \text{numerator}/\text{denominator} \).

- **Update State** (Line 16). The state \( \hat{X}_i(t) \) updates a step \( \alpha_i(t)d_i(t) \) along its last state.

Overall, all of the above DCG procedures are completely distributed and implemented through neighborhood message passing.

\begin{center}
\textbf{Algorithm 1: Distributed Conjugate Gradient (DCG) of} \( v_i \)
\end{center}

\begin{verbatim}
Input: \( \Omega_{(i,:)} \); \( b_i \);
Output: \( \hat{X}_i \);
2 \( \hat{X}_i(0) \leftarrow 0 \); \( d_i(0) \leftarrow 0 \); \( r_i(0) \leftarrow -b_i // Initialize
3
4 for iterations \( t \in \{ 1, \cdots, t_{\text{max}} \} \) do
5 \( r_i(t) = (\Omega\hat{X}(t-1))_i - b_i; // residual
6 if \( r(t)^T r(t) < \varepsilon \) then
7 \hspace{1cm} \text{break} // meets accuracy requirement
8 \hspace{1cm} d_i(t) = -r_i(t) + \frac{r(t)^T r(t)}{r(t(-1))^T r(t(-1))} d_i(t-1) // direction
9 \hspace{1cm} \alpha_i(t) = -\frac{d_i(t)^T r(t)}{d_i(t(-1))^T r(t(-1))} // step size
10 \hspace{1cm} \hat{X}_i(t) = \hat{X}_i(t-1) + \alpha_i(t)d_i(t) // state
11 \hspace{1cm} return \( \hat{X}_i(t) \).
12
13 \textbf{Function Synchronize\_Vector}(\( z_i \))
14 initialize \( z_i(0) \leftarrow [0_{(i-1)\times 1} \; z_i \; 0_{n-i\times 1}] \);
15 for iterations \( k \in \{ 0, \cdots, H \} \) do
16 exchange \( z_i(k) \) with \( \mathcal{N}_i \);
17 for each \( v_j \in \mathcal{N}_i \) do
18 \hspace{1cm} \( z_i(k+1) \leftarrow \text{merge}(z_i(k), z_j(k)) \);
19 return \( z_i \).
\end{verbatim}

C. Analysis of DCG

In each round, four vectors are obtained by synchronization, so it converges within \( 4HN \) rounds when \( \hat{A} \) is non-singular. In actual applications, there may exist measurement noise so that \( A \) and \( b \) are influenced and the constructed linear system may be unsolvable[9]. Let \( \bar{A} \) and \( \bar{b} \) denote the noisy matrices as:

\[
\bar{A} = A + \Delta A, \; \bar{b} = b + \Delta b,
\]  
(15)
where \( \Delta A \) and \( \Delta b \) are error matrices implying the noise. If the noisy matrix \( \overline{X} \) is still non-singular, DCG still converges to the neighborhood of \( X^* \).

**Lemma 1.** For a noisy linear system \( \overline{X}X = b \), DCG converges if the error matrix satisfies:

\[
||\Delta A|| < \lambda_{\min}(A).
\]

**Proof.** From (15), we can obtain:

\[
\overline{X} = A(I + A^{-1} \Delta A).
\]

Since \( ||A^{-1}\Delta A|| \leq ||A^{-1}|| ||\Delta A|| = \frac{||\Delta A||}{\lambda_{\min}(A)} \), the matrix \( \overline{X} \) must be non-singular if (16) holds. Thus DCG can converge to the neighborhood of \( \overline{X}^* \) shown as: \( \overline{X}^* = \overline{X}^{-1}b \).

IV. APPLICATIONS OF DCG

In this section, we investigate applying DCG to two actual scenarios.

A. The Least Square Problem

Linear equations arising from the era of engineering are usually over-determined. A typical scenario is distributed parameter estimation, where the observation equations are as:

\[
A_i x_i = b_i + \delta_i,
\]

where \( x_i \) is the desired parameter to be calculated and \( \delta_i \) is the component implying the measurement noise of \( b_i \). Due to the measurement noise, such linear equations usually do not have a solution that exactly meets the constraints. This problem can be formulated as follows. Suppose there is no solution to the linear system \( AX = b \) and \( AA^T \) is non-singular. Each agent \( v_i \) only knows the \( i \)th row of \( A \) and \( b \), which are denoted by \( A_{i} \) and \( b_{i} \), respectively. Design a distributed rule for each agent to update its state \( x_i \) so that \( x_i(t) \) converges to the unique solution of:

\[
A^TAX = A^Tb.
\]

Let \( \Omega = A^TA \) and \( \beta = A^Tb \). To solve the least square problem as in (19), each agent should be aware of the \( i \)th row of \( \Omega \) and \( \beta \) using \( A_{i} \) and \( b_{i} \).

Initially, \( v_i \) maintains \( A_{i} \). Then, \( v_i \) transmits \( A_{i} \) to \( N_i \) and receives \( A_{j} \) from \( N_j \), then it also knows the nonzero elements of the \( j \)th column \( A_{j} \), i.e., \( A_{j}^T \). Thus the nonzero elements of \( \Omega_{i} \) are calculated distributively as:

\[
\Omega_{ij} = \sum_{k=1}^{n} A_{ik}^T A_{kj}, \forall v_j \in N_i.
\]

Similarly, \( \beta_{i} \) can be calculated as:

\[
\beta_{ij} = \sum_{k=1}^{n} A_{ik}^T b_{kj}, j \in \{1, \ldots, d\}.
\]

Finally, the problem can be solved as \( x_i = DCG(\Omega_{i}, \beta_{i}) \). Therefore \( x \) solves the least squares problem in (19).

B. The Network Localization Problem

The geographical locations of nodes are fundamental information for many multi-agent applications[10], [11], [12]. Network localization techniques are usually adopted for calculating node locations in infrastructure-less scenarios[13], [14], which are formulated as follows. For a network of \( m + n \) agents in \( \mathbb{R}^d \), let \( V = A \cup F \) denote the entire node set. Nodes in \( A = \{v_1, \ldots, v_m\} \) are called anchor agents, whose locations \( P_A = \{p_1, \ldots, p_m\} \) are known. Nodes in \( F = \{v_{m+1}, \ldots, v_{m+n}\} \) are called free agents, whose locations \( P_F = \{p_{m+1}, \ldots, p_{m+n}\} \) are unknown. Each agent \( v_i \) can only sense the relative distance \( d_{ij} \) between \( v_i \) and any neighbor \( v_j \in N_i \). Each agent can exchange its estimated location \( p_i \) and distance measurements with neighbors. The network localization problem is to design a distributed protocol for each agent to update its location \( p_i \) so that it converges to \( p_i \).

To solve the network localization problem, we transform it to a linear system. First, each location \( p_i \) is represented as a linear combination of locations of neighbors:

\[
p_i = \sum_{v_j \in N_i} a_{ij} p_j,
\]

where \( a_{ij} \) are called barycentric coordinates. The calculation of barycentric coordinates involve only local distance measurements and the specific process is introduced by Diao et al. [2] in \( \mathbb{R}^2 \) and by Han et al. [15] in \( \mathbb{R}^3 \). Then after calculating the barycentric coordinates for each node, the agent locations can form a linear system:

\[
\begin{bmatrix}
P_A \\
P_F
\end{bmatrix}
= 
\begin{bmatrix}
I & 0 \\
B & C
\end{bmatrix}
\begin{bmatrix}
P_A \\
P_F
\end{bmatrix}.
\]

Writing \( I - C \) as \( M \), (24) can be reformulated as:

\[M_P = BP_A.\]

Considering that CG is used when the system matrix is positive definite [3]. Thus, we multiply \( M^T \) to both sides of (25):

\[M^T M_P = M^T B P_A.\]

Then, \( M^T M \) is positive definite. To apply DCG to distributed localization, the localization model in (26) is reformulated to \( \Omega \beta = \beta \), where \( \Omega = M^T M \in \mathbb{R}^{n \times n} \) and \( \beta = M^T B P_A \in \mathbb{R}^{n \times d} \).

Algorithm [2] shows the routine of DCG-Loc. For initialization, \( v_i \) needs to know the \( i \)th row of \( \Omega \) and \( \beta \) to invoke the general DCG. After constructing the local linear model by (22), \( v_i \) maintains \( A_{i} \), so it knows the nonzero elements of the \( i \)th row \( M_{i} \) by \( M_{i} = -A_{i} \) if \( v_j \in N_i \). Then, \( v_i \) transmits \( M_{i} \) to \( N_i \) and receives \( M_{j} \) from \( N_j \) (Line 3-6), so it also knows the nonzero elements of the \( i \)th column \( M_{i,j} \).
i.e., $M^T_i$. Thus the nonzero elements of $\Omega_{ij}$ are calculated distributively as:

$$\Omega_{ij} = \sum_{k=1}^n M^T_{ik} M_{kj}, \forall v_j \in N_i.$$  \hspace{1cm} (27)

To calculate $\beta_{(i,:)}$, an intermediate vector $\mu \in \mathbb{R}^{n \times d}$ implying $\text{BP}_A$ is introduced. $v_i$ locally calculates the $i$th row of $\mu$:

$$\mu_{i,:} = \sum_{v_a \in N_i \cap A} A_{ia} p_a, \hspace{1cm} (28)$$

where $N_i \cap A$ represents neighboring anchors. $\mu_{ij} = [0, 0]$ if no anchor is found in $N_i$. Then, $v_i$ sends $\mu_{i,:}$ to $N_i$ and receives $\mu_{j,:}$ of each $v_j \in N_i$ (Line 4). Thus, $\beta_{i,:}$ is calculated as:

$$\beta_{i,:} = \sum_{v_j \in N_i \cap A} M^T_{ij} \mu_{j,:},$$  \hspace{1cm} (29)

where $N_i \cap A$ represents the non-anchor barycentric neighbors. For convenience, each location $\hat{p}_i \in \mathbb{R}^{d \times 1}$ is decomposed to $[\beta_1, \ldots, \beta_d]$. The element $\beta_d$ is calculated by $\text{DCG}(\hat{A}_{i,:}, \beta_{d,:})$ (Line 10). Therefore, from the GBLL model in (27), $\hat{p}_i$ is calculated leveraging $\text{DCG-Loc}$, where communications only involve message passing with neighbors.

**Algorithm 2: Distributed Conjugate Gradient Localization (DCG-Loc) of $v_i$**

*Input*: neighbors: $N_i$; barycentric coordinates: $A_{i,:}$; 
*Output*: location: $\hat{p}_i$;

1. $M_{ij} \leftarrow -A_{ij}$; calculate $\mu_{i,:}$ as [28];
2. transmit $M_{ij}$ and $\mu_{i,:}$ to $v_j \in N_i$;
3. receive $M_{ji}$ and $\mu_{j,:}$ from $v_j \in N_i$;
4. calculate $\Omega_{ij}$ as [27]; calculate $\beta_{i,:}$ as [29];
5. return $\hat{p}_i \leftarrow [\text{DCG}(\Omega_{i,:}, \beta_{1,:}), \ldots, \text{DCG}(\Omega_{i,:}, \beta_{d,:})].$

**V. EVALUATION**

In this section, we evaluate the convergence speed between our proposed DCG algorithm and the representative Richardson iteration by counting the iteration rounds. Simulations are conducted using MATLAB R2020b in both $\mathbb{R}^2$ and $\mathbb{R}^3$. A network denoted by $\mathcal{G} = \{V, E\}$ is deployed. The number of anchors is set to $d + 1$, which is the minimum number of anchors required to uniquely localize a network in $\mathbb{R}^d$.

Fig. 1 and Fig. 2 shows the results in $\mathbb{R}^2$. DCG and Richardson iteration are adopted to solve the linear localization problem modeled by the network in Fig. 1. It is shown that both DCG and Richardson successfully converge to the ground truth within finite rounds of iterations. However, Fig. 2 shows that the Richardson iteration consumes $5 \times 10^5$ rounds while DCG only needs 550 rounds. From Fig. 3 and Fig. 4, similar results can be obtained in $\mathbb{R}^3$. Overall, DCG-Loc is shown to be faster than Richardson iteration about 1,000 times.

**VI. CONCLUSION**

In this paper, we proposed DCG to enable a network of $n$ agents to solve linear equations like $AX = b$ in fixed rounds of iterations. The DCG algorithm is presented with property analysis and two applications. Compared with traditional Richardson iteration, DCG shows 3 magnitudes faster convergence speed. In future work, we will consider reducing the communication burden that DCG requires.

**REFERENCES**

[1] Shaoshuai Mou, A Stephen Morse, Zhiyun Lin, Lili Wang, and Daniel Fullmer. A distributed algorithm for efficiently solving linear equations. In 2015 54th IEEE Conference on Decision and Control (CDC), pages 6791–6796. IEEE, 2015.
[2] Y. Diao, Z. Lin, and M. Fu. A barycentric coordinate based distributed localization algorithm for sensor networks. IEEE Transactions on Signal Processing, 62(18):4760–4771, 2014.
[3] Wolfgang Hackbusch. Iterative Solution of Large Sparse Systems of Equations, chapter 10.2, page 234. Applied Mathematical Sciences. Springer International Publishing, 2 edition, 2016.
[4] Charu C Aggarwal. Linear Algebra and Optimization for Machine Learning. Springer, 2020.
[5] Richard Barrett, Michael Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. Templates for the solution of linear systems: building blocks for iterative methods. SIAM, 1994.
[6] Yousef Saad. Iterative methods for sparse linear systems. SIAM, 2003.
[7] Leila Ismail and Rajeev Barua. Implementation and performance evaluation of a distributed conjugate gradient method in a cloud computing environment. Software: Practice and Experience, 43(3):281–304, 2013.
[8] Songcen Xu, Rodrigo C De Lameare, and H Vincent Poor. Distributed estimation over sensor networks based on distributed conjugate gradient strategies. IET Signal Processing, 10(3):291–301, 2016.
[9] Xu Fang, Xiaolei Li, and Lihua Xie. 3-d distributed localization with mixed local relative measurements. IEEE Transactions on Signal Processing, 68:5869–5881, 2020.
[10] Thien-Minh Nguyen, Zhirong Qiu, Thien Hoang Nguyen, Muqing Cao, and Lihua Xie. Persistently excited adaptive relative localization and time-varying formation of robot swarms. IEEE Transactions on Robotics, 36(2):553–560, 2020.
[11] T. Sun, Y. Wang, D. Li, Z. Gu, and J. Xu. Wcs: Weighted component stitching for sparse network localization. IEEE/ACM Transactions on Networking, 26(5):2242–2253, 2018.
[12] Y. Wang, T. Sun, G. Rao, and D. Li. Formation tracking in sparse airborne networks. *IEEE Journal on Selected Areas in Communications*, 36(9):2000–2014, 2018.

[13] H. Ping, Y. Wang, D. Li, and T. Sun. Flipping free conditions and their application in sparse network localization. *IEEE Transactions on Mobile Computing*, pages 1–1, 2020.

[14] Haodi Ping, Yongcai Wang, and Deying Li. Hgo: Hierarchical graph optimization for accurate, efficient, and robust network localization. In *Proceedings of the 29th International Conference on Computer Communications and Networks, ICCCN*, pages 1–9, 2020.

[15] T. Han, Z. Lin, R. Zheng, Z. Han, and H. Zhang. A barycentric coordinate based approach to three-dimensional distributed localization for wireless sensor networks. In *2017 13th IEEE International Conference on Control Automation (ICCA)*, pages 600–605, 2017.