Self-stabilizing Numerical Iterative Computation

Ezra N. Hoch[1], Danny Bickson[2] and Danny Dolev[1⋆]

[1] School of Computer Science and Engineering
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
[2] IBM Haifa Research Lab
Mount Carmel
Haifa 31905, Israel

Abstract. Many challenging tasks in sensor networks, including sensor calibration, ranking of nodes, monitoring, event region detection, collaborative filtering, collaborative signal processing, *etc.*, can be formulated as a problem of solving a linear system of equations. Several recent works propose different distributed algorithms for solving these problems, usually by using linear iterative numerical methods.
In this work, we extend the settings of the above approaches, by adding another dimension to the problem. Specifically, we are interested in *self-stabilizing* algorithms, that continuously run and converge to a solution from any initial state. This aspect of the problem is highly important due to the dynamic nature of the network and the frequent changes in the measured environment.
In this paper, we link together algorithms from two different domains. On the one hand, we use the rich linear algebra literature of linear iterative methods for solving systems of linear equations, which are naturally distributed with rapid convergence properties. On the other hand, we are interested in self-stabilizing algorithms, where the input to the computation is constantly changing, and we would like the algorithms to converge from any initial state. We propose a simple novel method called SS-Iterative as a self-stabilizing variant of the linear iterative methods. We prove that under mild conditions the self-stabilizing algorithm converges to a desired result. We further extend these results to handle the asynchronous case.
As a case study, we discuss the sensor calibration problem and provide simulation results to support the applicability of our approach.

1 Introduction

Many challenging tasks in sensor networks, for example distributed ranking algorithms of nodes and data items [3], collaborative filtering [1], localization [10], collaborative signal processing [12], region detection [9], etc.,

⋆ Part of the work was done while the author visited Cornell university. The work was funded in part by ISF.

can be formulated as a problem of solving a linear system of equations. Several recent works [10], [12],[9] propose different distributed algorithms for solving these problems, usually by linear iterative numerical methods.

In this work, we extend the settings of the above approaches by adding another dimension to the problem. Specifically, we are interested in *self-stabilizing* algorithms, that continuously run and converge to a solution from any initial state. This aspect of the problem is highly important due to the dynamic nature of the network and the frequent changes in the measured environment.

As a case study, we show that the calibration of local sensors' readings can be formulated as a linear system of equations $A\mathbf{x} = \mathbf{b}$, where $\mathbf{x}$ represents the calibrated output reading, $\mathbf{b}$ represents the local reading, and A represents a weighted communication graph. However, our work is general and can be applied to any problem that can be formulated as a distributed solution to a linear system of equations, including the previous works mentioned above.

Consider a distributed system of sensors measuring real-world data. Sensors are located in different areas; for example, the senors are spread throughout a building and they measure the temperature to adjust the heating or cooling. We would like the collected data to be as reliable as possible, reflecting closely the changing environmental conditions. One of the obstacles we face when designing algorithms that collect data from a sensor network are measurement errors. There are two main types of inaccuracies of sensors' measurements: noisy environment and sensing equipment which is not calibrated. It is desirable that sensors could execute a distributed algorithm for calibrating their environmental readings. In this setting sensors are allowed to communicate among themselves, using data from other nodes to affect their reported individual reading. Furthermore, we would like our calibration algorithm to have fault-tolerance properties. Specifically, we are interested in self-stabilizing algorithms [7] which converge to an optimal solution from any initial state. Observe that self-stabilization helps also in deploying the sensors. There is no need to explicitly synchronize the sensors, once enough of them are deployed and begin functioning the results will converge to the expected value.

The main challenge we have faced in this work, is that in the classical linear algebra literature, $\mathbf{b}$ is assumed to be constant. In our settings, the environment is constantly changing and the computed algorithm never terminates, leading to constantly changing values of $\mathbf{b}$. In this paper, we ask the following question: "Is it possible to devise a self-stabilizing numerical iterative method?" We answer affirmatively, and show that under minor conditions it is possible to devise a self-stabilizing algorithm that solves a dynamic system of linear equations, where the input to the system is constantly changing.

To the best of our knowledge, this is the first work tackling this challenging problem. We believe that our approach can have numerous applications in the field of distributed self-stabilizing computation.

Other works discuss fault tolerance aspects of distributed computation. For example, overcoming faults in sensors by averaging the input was investigated in [11] providing a centralized algorithm. Quantifying faulty nodes' effect on the system's output is discussed in [8] and [5]. These papers consider bounded input paths and their effect on the stability of the output. In [6] infinite input paths are considered under the assumption that only specific sensors' input may change. All three papers consider discrete input values, as opposed to a continuous set of input values discussed in this paper.

The paper is constructed as follows. Section 2 defines the model and problem definition. Section 3 presents our novel algorithm SS-Iterative. Section 4 analyzes our algorithm and gives bounds on the convergence rate. Section 5 presents experimental results of running SS-Iterative using sample topologies. Section 6 extends our construction to the asynchronous case. We conclude in Section 7.

2 Model and Problem Definition

We model the sensor calibration problem as follows. Given a directed communication graph $G = (V, E)$, V is the set of nodes $V = \{p_1, \ldots, p_n\}$, E is the set of weighted edges connecting them (weights can be negative) and $N(p_i)$ denotes p_i's neighbors. Edge weights are used to model the directional dependence between nodes' outputs; *i.e.,* if $w_{p_i,p_j} = 0$ then there is no edge between p_i and p_j and their output is not directly dependent on each other. In addition, we require a non-zero self connected edge, $w_{p_i,p_i} \neq 0$, which represents the weight of p_i's own input.

Initially, we assume a synchronous system: during a single round of communication, any pair of connected nodes may send a single message on each directed edge. Each round r, each node p_i has a scalar input value $I_{p_i}(r)$, which represents the local reading of the sensor.[3] In addition, p_i outputs its output value, which is denoted by $O_{p_i}(r)$; both inputs and outputs are from the domain of real numbers. Denote by $I(r)$ the input

[3] For simplicity of notations we use scalar variables in the paper. An extension to the vector case (where each sensor measures a set of measurements) is immediate.

vector of the entire system at round r, and by $O(r)$ the output vector of the system at the end of round r. In Section 6 we relax the assumption of synchronous rounds and provide a variant of the algorithm which works in asynchronous settings.

The schematic operation of each node p_i at round r is composed of the following steps: (a) read the value of $I_{p_i}(r)$; (b) send messages; (c) receive messages; and (d) do some processing and output $O_{p_i}(r)$. Then a new round is started, and the nodes continue so forever.

Definition 1. *A* **configuration** $\mathcal{C}$ *of the system at round* r *consists of the state of each node prior to performing any operation at round* r*; this configuration is denoted by* $\mathcal{C}(r)$*.*

Definition 2. *An* **input sequence** $\mathcal{I}$ *of length* ℓ *is a list of* ℓ *vectors such that each* $\mathbf{v} \in \mathcal{I}$ *is a possible input vector of the system (*i.e., $\mathbf{v} \in \mathfrak{D}$*, the domain of allowed values). An* **output sequence** $\mathcal{O}$ *of length* ℓ *is a list such that each* $\mathbf{v} \in \mathcal{O}$ *can potentially be an output vector of the system.*

Definition 3. *A step from configuration* $\mathcal{C}$ *to configuration* $\mathcal{C}'$ *on input vector* $\mathbf{v}$ *is* **legal** *if* $\mathcal{C}'$ *is reached from* $\mathcal{C}$ *by the system when having* $\mathbf{v}$ *as the input vector.* $\mathbf{u}$ *is* **produced** *by a legal step if* $\mathbf{u}$ *is the output vector of the system resulting from such a legal step.*

Definition 4. *A* **run** *of a system on input sequence* $\mathcal{I} = \{\mathbf{v}(1), \ldots, \mathbf{v}(\ell)\}$ *starting from configuration* $\mathcal{C}(r)$ *is the sequence* $\mathcal{C}(r), O(r), \mathcal{C}(r+1), O(r+1), \ldots$ *s.t. for any* $i \geq 0$*: the step from* $\mathcal{C}(r+i)$ *to* $\mathcal{C}(r+i+1)$ *on input vector* $\mathbf{v}(i+1)$ *is legal, and* $O(r+i)$ *is produced by that legal step. The system is said to produce the output sequence* $\mathcal{O} = \{O(r), \ldots, O(r+\ell-1)\}$*.*

In the special case when the sensor observations (the input to the system) are fixed, the output decision of the sensors should converge to a solution that preserves the linear relations among node inputs and outputs. More formally, consider an input sequence $\mathcal{I}$ of identical input vectors; *i.e.,* $\mathcal{I} = \{\mathbf{v}, \mathbf{v}, \mathbf{v}, \ldots\}$. It is desired that for such an input a run from any configuration $\mathcal{C}$ on $\mathcal{I}$ would end up producing an output sequence $\mathcal{O} = \{\mathbf{u}(1), \mathbf{u}(2), \ldots\}$ such that $||\mathbf{u}(i) - \mathbf{u}|| \to 0$ as $i \to \infty$, for a $\mathbf{u}$ that solves the following linear system of equations:

$$\mathbf{u}_i = w_{p_i,p_i} \cdot \mathbf{v}_i + \sum_{p_j \in N(p_i)} w_{p_i,p_j} \cdot \mathbf{u}_j \ . \qquad (1)$$

We assume that the above equations are uniquely solvable, denoting $\mathbf{u}$ as the solution to $\mathbf{v}$.

One of the most efficient distributed approaches for solving a set of linear equations of the type $A\mathbf{x} = \mathbf{b}$ is by using linear iterative algorithms. Unlike Gaussian elimination, which has a cost of $O(n^3)$, where n is the number of variables, an iterative algorithm usually solves a system of linear equations in time of $O(n^2 r,)$ where r is the number of iterations, which is typically logarithmic in n. These algorithms are naturally distributed and work well in asynchronous settings. Furthermore, when converging, the algorithms converge to a solution from *any* initial state. An excellent survey of such methods is found in [2].

The main novel contribution of this paper is in analyzing the self-stabilizing properties of algorithms from the linear iterative methods domain. In a practical setting, it is highly unreasonable to assume that sensor readings do not change over time. However, it is reasonable to assume that at steady state the change in sensor readings is bounded. Informally, in this work we show that once the input readings are bounded, the output solution is bounded as well. This useful observation enables us to tie together numerical iterative methods and dynamically changing environments in a self-stabilizing manner.

The following definition bounds the change in sensor observations:

Definition 5. *An input sequence $\mathcal{I} = \{\mathbf{v}(1), \mathbf{v}(2), \ldots, \mathbf{v}(\ell)\}$ is δ-**bounded** around $\mathbf{v}$ if for every i, $1 \le i \le \ell$, it holds that $||\mathbf{v}(i) - \mathbf{v}||_\infty \le \delta$.*[4]

Definition 5 states that a sequence $\mathcal{I}$ is δ-bounded if all the vectors in $\mathcal{I}$ are bounded within an n dimensional hypercube with an edge 2δ, centered around a point $\mathbf{v}$. We note that once changes in the input are not bounded, then no efficient algorithm (especially in a network that is sparsely connected) can calculate the output fast enough. For example, if the diameter of the communication graph is $\mathcal{D}$, for some system of equations it would take at least $\mathcal{D}$ rounds for the information exchange for input readings at one side of the network to propagate to the other side of the network.

Definition 6. *Let $\mathcal{I}$ be an input sequence that is δ-bounded around $\mathbf{v}$ and let $\mathbf{u}$ be the solution to input $\mathbf{v}$. A run from configuration $\mathcal{C}$ on input sequence $\mathcal{I}$ ϵ-**converges** to its solution if the produced output sequence $\mathcal{O} = \{\mathbf{u}(1), \mathbf{u}(2), \ldots \mathbf{u}(\Delta t)\}$ satisfies that $||\mathbf{u}(\Delta t) - \mathbf{u})||_\infty \le \epsilon(\Delta t, \delta, \mathcal{C})$; where ϵ is a function of $\Delta t, \delta$ and $\mathcal{C}$.*

[4] $||\mathbf{x}||_\infty = \max_i\{|\mathbf{x}_i|\}$.

Definition 6 requires that if - starting from configuration $\mathcal{C}$ - the inputs are in an n dimensional hypercube of radius δ around $\mathbf{v}$ then the output at time Δt is bounded within some n dimensional hypercube around $\mathbf{u}$ with radius $\epsilon(\Delta t, \delta, \mathcal{C})$. We aim at an $\epsilon(\Delta t, \delta, \mathcal{C})$ that decreases as Δt increases, as long as the inputs are bounded by the same $\mathbf{v}$-centered, δ-radius hypercube. Clearly, for $\delta > 0$, $\epsilon(\Delta t, \delta, \mathcal{C}) > 0$ for any Δt. That is, there is some minimal radius $\delta' > 0$ around $\mathbf{u}$ s.t. we cannot ensure a tighter bound.

The above definition considers a single initial configuration, and a single input sequence $\mathcal{I}$. We are interested in an algorithm that works for all initial configurations and all input sequences.

Definition 7. *An algorithm $\mathcal{A}$ ϵ-**converges** for δ-bounded input sequence $\mathcal{I}$ if every run (from any configuration) on $\mathcal{I}$, ϵ-converges to its solution. An algorithm $\mathcal{A}$ ϵ-**always converges** if for every δ-bounded input sequence $\mathcal{I}$, $\mathcal{A}$ ϵ-converges.*

Definition 7 formally defines the problem at hand, as an algorithm $\mathcal{A}$ that *always converges* has the desired self-stabilizing property: for any system state, once the sensors' readings changes are bounded, the change in output of the entire system is bounded as well.

Our goal is to find an algorithm $\mathcal{A}$ that is ϵ-always converging for a provably "good" ϵ. Moreover, we aim at having $\mathcal{A}$ efficient also in its message complexity and simplicity of code, allowing lightweight sensors to actually implement it.

3 Our Proposed Solution

An equivalent formulation of the update rule Eq. (1) is

$$\mathbf{u}_i = w_{p_i,p_i} \cdot \mathbf{v}_i + \sum_{j \neq i} w_{p_i,p_j} \cdot \mathbf{u}_j \ .$$

The above equation states a condition on p_i's output, in regard to p_i's inputs and p_i's neighbors' output. Thus, it encapsulates the requirement that different nodes influence each other's reported readings, while taking into consideration their local readings as well.

Since $w_{p_i,p_i} \neq 0$, the above equation can be stated as:

$$\frac{1}{w_{p_i,p_i}} \mathbf{u}_i - \sum_{j \neq i} \frac{w_{p_i,p_j}}{w_{p_i,p_i}} \cdot \mathbf{u}_j = \mathbf{v}_i \ .$$

By denoting $w_{i,j} = -\frac{w_{p_i,p_j}}{w_{p_i,p_i}}$ (for $i \neq j$) and $w_{i,i} = \frac{1}{w_{p_i,p_i}}$ we get:

$$\sum_j w_{i,j} \cdot \mathbf{u}_j = \mathbf{v}_i \ . \tag{2}$$

Let W be the matrix that has $w_{i,j}$ as entries, Eq. (2) can be written in linear algebra notation, (s.t. it applies to all nodes simultaneously):

$$W\mathbf{u} = \mathbf{v} \ . \tag{3}$$

If we consider a non-self-stabilizing system in which the inputs do not change (that is, the input is fixed to $\mathbf{v}$), then Eq. (3) can be seen as $A\mathbf{x} = \mathbf{b}$, where A and $\mathbf{b}$ are given. In such a case, we are interested in finding the value of $\mathbf{x}$, which is a vector of n unknown variables. However, we are interested in the case where $\mathbf{v}$ changes over time, and thus Eq. (3) does not describe the problem properly, but rather helps in understanding the motivation for our solution.

We use a modified update rule (relative to Eq. (1)):

$$O_{p_i}(r+1) = w_{p_i,p_i} \cdot I_{p_i}(r+1) + \sum_{j \neq i} w_{p_i,p_j} \cdot O_{p_j}(r) \ . \tag{4}$$

Clearly, for the case of $\delta = 0$, a 0-bounded input sequence $\mathcal{I}$, if $(O_{p_i}(r+1) - O_{p_i}(r)) \longrightarrow 0$ as $r \rightarrow \infty$ then Eq. (4) converges to the solution of Eq. (1). Thus, if the update rule of Eq. (4) is executed simultaneously by all nodes, and for all of the nodes $(O_{p_i}(r+1) - O_{p_i}(r)) \longrightarrow 0$, then it also solves Eq. (3). That is, if each node locally executes Eq. (4) then the global solution is reached. This observation motivates algorithm SS-Iterative in Figure 1.

Remark 1. In SS-Iterative there is no notion of the "current round number r". That is, p_i reads and writes to the variables I_{p_i} and O_{p_i} without being "aware" of r. When we discuss the algorithm "from the outside", we will consider $I_{p_i}(r)$ and $O_{p_i}(r)$ instead of just I_{p_i}, O_{p_i}.

Consider p_i is running at round $r+1$. When p_i performs Line `03`, it sends the value of O_{p_i}. The last time O_{p_i} was updated was at Line `04` and Line `06` of round r. Thus, the value sent by p_i at round $r+1$ is actually $O_{p_i}(r)$. Therefore, the values received from p_j by p_i and used to update $O_{p_i}(r+1)$ are $O_{p_j}(r)$. However, the value read by p_i in Line `04` is the value of $I_{p_i}(r+1)$. Concluding that p_i updates $O_{p_i}(r+1)$ exactly according to Eq. (4).

Remark 2. Each node p_i must know the values of w_{p_i,p_j} as "part of the code". Thus, these values cannot be subject to transient faults.

Algorithm SS-Iterative

```
01: Each round do:                                    /* executed on node $p_i$ */

        /* send current value of $O_{p_i}$ to all neighbors */
02:     for each $p_j \in N(p_i)$
03:         send $O_{p_i}$ to $p_j$;

        /* update $O_{p_i}$ according to values sent by neighbors */
04:     set $O_{p_i} := w_{p_i,p_i} \cdot I_{p_i}$;
05:     for each value $O_{p_j}$ received:
06:         update $O_{p_i} := O_{p_i} + w_{p_i,p_j} \cdot O_{p_j}$;

07: od.
```

Fig. 1. A self-stabilizing iterative algorithm.

4 Analysis of SS-Iterative

[2] shows that the update rule Eq. (4) can be written in linear algebra form as

$$O(r+1) = AI(r+1) + BO(r) \ , \tag{5}$$

where A is a diagonal matrix with w_{p_i,p_i} in the main diagonal, and $B_{ij} = w_{p_i,p_j}$ for $i \neq j$

$$A \triangleq (diag\{W\})^{-1} \quad , \quad B \triangleq -(AW - I_{n\times n}) \tag{6}$$

where $I_{n\times n}$ is the identity matrix. Using this update rule to solve a set of linear equations iteratively is known as the Jacobi algorithm.

As noted in Section 2, when the input sequence is constant (*i.e.*, $I(r) = \mathbf{v}$ for all r) the iterative execution of the above equations converges to $\mathbf{u} = A\mathbf{v} + B\mathbf{u}$, which is the same as $\mathbf{u} = W^{-1}\mathbf{v}$, thus solving Eq. (3). Following, we analyze the result of iteratively applying these equations for δ-bounded input sequences.

Let $\mathcal{I}$ be an input sequence of length ℓ that is δ-bounded around vector $\mathbf{v}$. That is, $\mathcal{I} = I(r), I(r+1), \ldots, I(r+\ell-1)$ for some round r. Note that SS-Iterative saves a single scalar variable at each node, and thus the configuration of round $r+1$ can be defined by the value of $O(r)$ at round r. Consider SS-Iterative's run, starting from an arbitrary configuration at round r. We aim at showing that $O(r+\Delta t)$ is bounded by a hypercube centered at $\mathbf{u}$. Denote by $\mathbf{c}(\Delta t) \triangleq O(r+\Delta t) - \mathbf{u}$. If we show that $||\mathbf{c}(\Delta t)||_\infty$ is bounded (as Δt increases), then $O(r+\Delta t)$ is within a bounded hypercube centered at $\mathbf{u}$. Consider $\mathbf{c}(1)$:

$$\begin{aligned}\mathbf{c}(1) &= O(r+1) - \mathbf{u} \\ &= AI(r+1) + BO(r) - (A\mathbf{v} + B\mathbf{u}) \\ &= A(I(r+1) - \mathbf{v}) + B(O(r) - \mathbf{u}) \\ &= A(I(r+1) - \mathbf{v}) + B\mathbf{c}(0)\ . \end{aligned} \tag{7}$$

Since $\mathcal{I}$ is a δ-bounded input sequence around $\mathbf{v}$, each $I(r + \Delta t)$ can be denoted as $\mathbf{v} + \mathcal{D}(r + \Delta t)$ s.t. $\mathcal{D}(r + \Delta t) \in \mathbb{R}^n$ is a vector, and $||\mathcal{D}(r + \Delta t)||_\infty \leq \delta$. That is, $\mathcal{D}(r + \Delta t) = I(r + \Delta t) - \mathbf{v}$.

Claim. At round $r + \Delta t$, it holds that $\mathbf{c}(\Delta t) = \sum_{j=0}^{\Delta t-1} B^j A\mathcal{D}(r + \Delta t - j) + B^{\Delta t}\mathbf{c}(0)$.

Proof. Proof by induction. The base of the induction was shown for $\mathbf{c}(1)$; see Eq. (7). Assume that the claim holds for $\Delta t = k$. Thus, $\mathbf{c}(k) = \sum_{j=0}^{k-1} B^j A\mathcal{D}(r + k - j) + B^k\mathbf{c}(0)$. By the update rule in Eq. (5), we have that $O(r + k + 1) = AI(r + k + 1) + BO(r + k)$. Combining the two equations implies

$$\begin{aligned}\mathbf{c}(k+1) &= O(r+k+1) - \mathbf{u} \\ &= AI(r+k+1) + BO(r+k) - (A\mathbf{v} + B\mathbf{u}) \\ &= A(I(r+k+1) - \mathbf{v}) + B(O(r+k) - \mathbf{u}) \\ &= A\mathcal{D}(r+k+1) + B\mathbf{c}(k) \\ &= A\mathcal{D}(r+k+1) + \sum_{j=0}^{k-1} B^{j+1}A\mathcal{D}(r+k-j) + B^{k+1}\mathbf{c}(0) \\ &= A\mathcal{D}(r+k+1) + \sum_{j=1}^{k} B^{j}A\mathcal{D}(r+k+1-j) + B^{k+1}\mathbf{c}(0) \\ &= \sum_{j=0}^{k} B^{j}A\mathcal{D}(r+k+1-j) + B^{k+1}\mathbf{c}(0)\ . \end{aligned}$$

Thus, if the claim holds for $\Delta t = k$ it also holds for $\Delta t = k + 1$; and we have that the claim holds for all $\Delta t \geq 0$. □

Definition 8. *A matrix $M_{n\times n}$ is* **diagonally dominant** *if $|M_{ii}| > \Sigma_{j\neq i}^n |M_{ij}|$. A matrix $M_{n\times n}$ is* **normalized** *diagonally dominant (normalized, for short) if M is diagonally dominant, and $|M_{ii}| \geq 1$.*

Lemma 1. *For a normalized diagonally dominant matrix W, it holds that $||A||_\infty \leq 1$ and $||B||_\infty < 1$, where A, B are defined in Eq. (6) and $||A||_\infty \triangleq \max_{x\neq 0} \frac{||Ax||_\infty}{||x||_\infty}$.*

Proof. A is zero except for its main diagonal for which $A_{i,i} = w_{p_i,p_i} = \frac{1}{w_{i,i}}$. Since $|W_{ii}| \geq 1$, we have that $|A_{i,i}| \leq 1$. Thus, it holds that $||A\mathbf{x}||_\infty \leq ||\mathbf{x}||_\infty$. Furthermore, $\max_{\mathbf{x}\neq 0} \frac{||A\mathbf{x}||_\infty}{||\mathbf{x}||_\infty} \leq 1$, *i.e.*, $||A||_\infty \leq 1$. Regarding B, $B_{i,j} = w_{p_i,p_j}$ for $i \neq j$ and 0 for $i = j$. Since W is assumed to be normalized diagonally dominant, we have that $\sum_{j\neq i} |W_{i,j}| < |W_{i,i}|$, thus $\sum_{j\neq i} |w_{p_i,p_j}| < 1$. Therefore, $\sum_j |B_{i,j}| = \sum_{j\neq i} |w_{p_i,p_j}| < 1$ for all i. In total, for any $\mathbf{x}$ we have $||B\mathbf{x}||_\infty < ||\mathbf{x}||_\infty$, leading to $||B||_\infty < 1$. □

If W is a diagonally dominant matrix then node p_i's own input effects p_i's output more than the sum of all of p_i's neighbors outputs. That is, the weight of p_i's input is at least the sum of weights of p_i's neighbors outputs.

Theorem 1. *Given a normalized diagonally dominant and invertible W, there are constants c_1, c_2, where $c_1 > 0$, and $1 > c_2 > 0$, such that* SS-ITERATIVE *ϵ-always converges with $\epsilon(\Delta t, \delta, \mathcal{C}) = \delta \cdot c_1 + (c_2)^{\Delta t} \cdot ||O(r) - \mathbf{u}||_\infty$.*

Proof. By Lemma 1 it holds that $||A||_\infty \leq 1$ and $||B||_\infty < 1$. Consider a δ-bounded input sequence $\mathcal{I}$ around $\mathbf{v}$, and SS-ITERATIVE's run starting from an arbitrary state $O(r)$. We are interested in the behavior of $||\mathbf{c}(\Delta t)||_\infty$:

$$\begin{aligned}
||\mathbf{c}(\Delta t)||_\infty &= \left|\left| \sum_{j=0}^{\Delta t-1} B^j A \mathcal{D}(r + \Delta t - j) + B^{\Delta t}\mathbf{c}(0) \right|\right|_\infty \\
&\leq \left|\left| \sum_{j=0}^{\Delta t-1} B^j A \mathcal{D}(r + \Delta t - j) \right|\right|_\infty + ||B^{\Delta t}\mathbf{c}(0)||_\infty \\
&\leq \sum_{j=0}^{\Delta t-1} ||B||_\infty^j \, ||A\mathcal{D}(r + \Delta t - j)||_\infty + ||B||_\infty^{\Delta t} \, ||\mathbf{c}(0)||_\infty \\
&\leq \delta \cdot ||A||_\infty \sum_{j=0}^{\Delta t-1} ||B||_\infty^j + ||B||_\infty^{\Delta t} \, ||\mathbf{c}(0)||_\infty \\
&= \delta \cdot ||A||_\infty \frac{1 - ||B||_\infty^{\Delta t}}{1 - ||B||_\infty} + ||B||_\infty^{\Delta t} \, ||\mathbf{c}(0)||_\infty \ . \qquad (8)
\end{aligned}$$

For an input sequence $\mathcal{I}$ that is δ-bounded around $\mathbf{v}$, denote by $\mathbf{u}$ the solution to the original system of equations $W\mathbf{u} = \mathbf{v}$. By Eq. (8),

$$||O(r + \Delta t) - \mathbf{u}||_\infty \leq \delta \cdot ||A||_\infty \frac{1 - ||B||_\infty^{\Delta t}}{1 - ||B||_\infty} + ||B||_\infty^{\Delta t} \, ||\mathbf{c}(0)||_\infty .$$

Since $||B||_\infty < 1$, we have that $\frac{1-||B||_\infty^{\Delta t}}{1-||B||_\infty} \leq \frac{1}{1-||B||_\infty}$ and by setting $c_1 = \frac{||A||_\infty}{1-||B||_\infty}$ it holds that $||A||_\infty \frac{1-||B||_\infty^{\Delta t}}{1-||B||_\infty} \leq c_1$. By setting $c_2 = ||B||_\infty$ and recalling that $\mathbf{c}(0) = O(r) - \mathbf{u}$ we are done. $\square$

Theorem 1 states sufficient conditions s.t. SS-Iterative ϵ-always converges. Moreover, the algorithm SS-Iterative is lightweight, as it requires nodes to send only a single value to every neighbor on each round.

5 Experimental Results

For illustrating the behavior of our proposed algorithm, we have simulated SS-Iterative using two sample topologies of one hundred nodes. Figure 2 depicts a circular topology where each node is connected to its left and right neighbors. Figure 3 shows a random unit disc graph, where nodes are randomly spread on a plane, and each node is connected to the nodes that are within a distance of 1. The X-axis shows the number of iterations, and the Y-axis shows the value of δ. Area colors in the heatmap depict the average of the following procedure: randomly select a vector $\mathbf{v}$ and a δ-bounded sequence around $\mathbf{v}$, run the simulation for the randomly selected values and return the L_∞ distance between the last output vector and $\mathbf{u}$ (calculated as $\mathbf{u} = W^{-1}\mathbf{v}$). The heatmap uses a $\log\log$ scale. Both graphs clearly show that as δ decreases and the number of iterations increases, the output of SS-Iterative converges to be bounded by a small hypercube around $\mathbf{u}$.

Note that the unit disc weighted topology matrix is characterized by $||A||_\infty = 0.02, ||B||_\infty = 0.97$ while the circle graph is characterized by $||A||_\infty = 0.33, ||B||_\infty = 0.66$. As expected, using unit disc topology requires a larger number of iterations for convergence (depends on $||B||_\infty$). In addition, in the unit disc topology the value of δ has a lesser effect on the convergence, due to the value of $||A||_\infty$, which affects the minimal radius around the output. Since $||A||_\infty$ is smaller in the unit disc topology, increasing δ does not significantly affect the convergence.

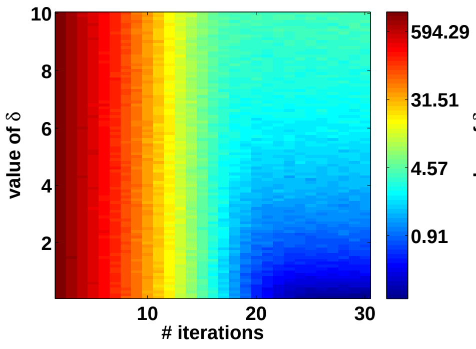

Fig. 2. Sim. of a Circle graph.

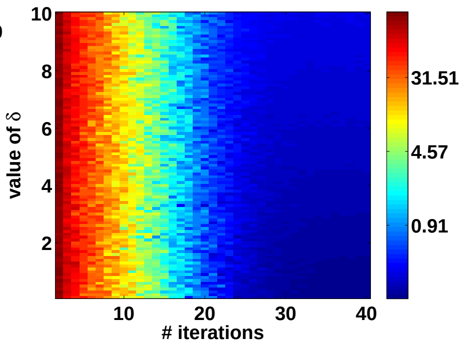

Fig. 3. Sim. of a Unit-Disc graph.

6 Extension to the asynchronous model

Our second novel contribution is in extending our model to support asynchronous communications. In a large sensor network, it is unreasonable to assume that the sensors operate in synchronous rounds. Furthermore, as known from the linear iterative algorithms literature, algorithms usually converge in less asynchronous rounds (when compared to synchronous rounds).

When considering the asynchronous model, it is more convenient to discuss shared-memory as means of communication.[5] Thus, assume that for each directed edge between p_i, p_j there is a read-write register R_{p_i,p_j} that is written by p_i and read by p_j.

An asynchronous run is an infinite sequence of configurations $\mathcal{C}_0 \rightarrow \mathcal{C}_1 \rightarrow \ldots$ such that some process p performs an atomic step between configuration $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$. An atomic step consists of reading or writing from a single register. Notice that in the current model a configuration consists of all of the registers and of the local variables at the different nodes.

In this section we again prove that starting from an arbitrary configuration, when the inputs are bounded, the outputs are bounded as well. We consider each configuration $\mathcal{C}_r$ to be assigned a vector input $I(r)$ such that if node p_i reads the input when performing an atomic step on $\mathcal{C}_r$ it reads the value of $I_{p_i}(r)$. Equivalently, the output vector of configuration $\mathcal{C}_r$ is $O(r)$.

[5] In [7] it is shown how to convert an algorithm based on shared-memory to a message-passing algorithm with links of bounded capacity.

Figure 4 outlines Async-SS-Iterative which is a direct translation of SS-Iterative to the shared-memory model.

```
Algorithm Async-SS-Iterative
01: Forever do:                                    /* executed on node $p_i$ */

        /* write current value of $O_{p_i}$ to all neighbors */
02:     for each $p_j \in N(p_i)$
03:         write $O_{p_i}$ to $R_{p_i,p_j}$;

        /* update $O_{p_i}$ according to values of neighbors */
04:     set $O_{p_i} := w_{p_i,p_i} \cdot I_{p_i}$;
05:     for each $p_j \in N(p_i)$:
06:         read $R_{p_j,p_i}$ into temp;
07:         update $O_{p_i} := O_{p_i} + w_{p_i,p_j} \cdot temp$;

08: od.
```

Fig. 4. A self-stabilizing iterative algorithm for asynchronous networks.

Async-SS-Iterative consists of two phases: in the first, the previous value of O_{p_i} is written to all its neighbors. In the second phase p_i calculates its new value of O_{p_i} by reading the registers of all its neighbors.

We consider only "fair" runs, in which each node performs an atomic step infinitely many times. Thus, each node performs both phases infinitely many times. A round is defined to be the shortest prefix of a run such that each node has performed lines 02-07 in the algorithm. We number each atomic step and each round. Note that a round consists of many atomic steps.

We model a fair run as follows. Each node p_i performs infinitely many atomic steps, and participates in infinitely many rounds. Notice that the registers p_i reads in round $k+1$ have all been last written to, no earlier than during round k. Since a round consists of each node performing all the steps in the algorithm, each node p_i manages to read all of its neighboring registers and write to all of its neighboring registers every round. Thus, there is some atomic step r (during round $k+1$) such that:

$$O_{p_i}(r) = w_{p_i,p_i} \cdot I_{p_i}(r') + \sum_{j \neq i} w_{p_i,p_j} \cdot O_{p_j}(r'_j) \ ,$$

where r', r'_j (for all $p_j \neq p_i$) are smaller than r and are from at least round k.

Let $\mathbf{u}$ be such that $\mathbf{u} = A\mathbf{v} + B\mathbf{u}$, and let the inputs be from a δ-bounded input sequence around $\mathbf{v}$. Denote $\mathbf{c}(r) = O(r) - \mathbf{u}$ and $z = \max_i |\mathbf{c}_{p_i}(0)|$.

Theorem 2. *Given a normalized diagonally dominant and invertible W, and while considering only fair runs, there are constants c_1, c_2, where $c_1 > 0$, and $1 > c_2 > 0$, such that* Async-SS-Iterative *ϵ-always converges with $\epsilon(\Delta t, \delta, \mathcal{C}) = \delta \cdot c_1 + (c_2)^{\Delta t} \cdot z$; where Δt counts the asynchronous rounds of a fair run.*

Proof. Notice that if p_i did not perform the rth atomic step then $O_{p_i}(r) = O_{p_i}(r-1)$ and therefore $\mathbf{c}_{p_i}(r) = \mathbf{c}_{p_i}(r-1)$. Consider the value of $\mathbf{c}_{p_i}(r)$ when p_i did perform the rth atomic step (during round $k+1$).

$$\begin{aligned}
\mathbf{c}_{p_i}(r) = O_{p_i}(r) - \mathbf{u}_{p_i}& \\
&= w_{p_i,p_i} \cdot I_{p_i}(r') + \sum_{j \neq i} w_{p_i,p_j} \cdot O_{p_j}(r'_j) - w_{p_i,p_i} \cdot \mathbf{v}_{p_i} - \sum_{j \neq i} w_{p_i,p_j} \cdot \mathbf{u}_{p_i} \\
&= w_{p_i,p_i} \cdot (I_{p_i}(r') - \mathbf{v}_{p_i}) + \sum_{j \neq i} w_{p_i,p_j} \cdot (O_{p_j}(r'_j) - \mathbf{u}_{p_i}) \\
&= w_{p_i,p_i} \cdot (I_{p_i}(r') - \mathbf{v}_{p_i}) + \sum_{j \neq i} w_{p_i,p_j} \cdot \mathbf{c}_{p_j}(r'_j) \ ,
\end{aligned}$$

where r' and the different r'_j are smaller than r and are all from round k or round $k+1$.

By using Lemma 1 we get:

$$\begin{aligned}
|\mathbf{c}_{p_i}(r)| &\leq |w_{p_i,p_i} \cdot (I_{p_i}(r') - \mathbf{v}_{p_i})| + \max_{p_j} |c_{p_j}(r'_j)| \sum_{j \neq i} |w_{p_i,p_j}| \\
&\leq |w_{p_i,p_i} \cdot (I_{p_i}(r') - \mathbf{v}_{p_i})| + ||B||_\infty \, |\mathbf{c}_{p_{max}}(r_{max})| \\
&\leq \delta + ||B||_\infty \, |\mathbf{c}_{p_{max}}(r_{max})| \ ,
\end{aligned}$$

for some p_{max} and $r_{max} \leq r$ that is from round k or $k+1$.

Therefore, for any p_i during round $k+1$ there is a list of length $\ell \geq k$ of nodes $p_1, p_2, \ldots, p_\ell$ and a sequence of length ℓ of atomic steps

$r_1 > r_2 > \cdots > r_\ell = 0$, such that

$$\begin{aligned}
|\mathbf{c}_{p_i}(r)| &\leq \delta + ||B||_\infty |\mathbf{c}_{p_1}(r_1)| \\
&\leq \delta + ||B||_\infty (\delta + ||B||_\infty |\mathbf{c}_{p_2}(r_2)|) \\
&= \delta \cdot (1 + ||B||_\infty) + ||B||_\infty^2 |\mathbf{c}_{p_2}(r_2)| \\
&\leq \delta \cdot \sum_{z=0}^{\ell-1} ||B||_\infty^z + ||B||_\infty^\ell |\mathbf{c}_{p_\ell}(r_\ell)| \\
&= \delta \cdot \frac{1 - ||B||_\infty^\ell}{1 - ||B||_\infty} + ||B||_\infty^\ell |\mathbf{c}_{p_\ell}(0)| \,.
\end{aligned}$$

Denote by $c_1 \triangleq \frac{1}{1-||B||_\infty}$, and $c_2 \triangleq ||B||_\infty$. We have that for node p_i performing the rth atomic step during round k it holds that $|\mathbf{c}_{p_i}(r)| \leq \delta \cdot c_1 + c_2^\ell \cdot z \leq \delta \cdot c_1 + c_2^k \cdot z$. □

In fair runs, there are infinitely many rounds k, thus, as l and r go to infinity, we have that $||O(r)||_\infty$ is bounded by a hypercube of length $\delta \cdot c_1$ around $\mathbf{u}$.

7 Discussion

We have shown that the algorithm SS-Iterative is a modification of the Jacobi iterative method to solve a set of equations $A\mathbf{x} = \mathbf{b}$, where A is given and $\mathbf{b}$ is dynamically changing but bounded. Moreover, Theorem 1 is a generalization of previous analysis of Jacobi's convergence. Our motivation for SS-Iterative originates from the sensor calibration problem where sensors need to calibrate their noisy readings. Unlike previous approaches to this problem, we assume a dynamic system with an infinite execution of the algorithm. In this setting the readings of the sensors continuously change. Under the assumption that the readings' changes are bounded, we have shown that the calibrated output is bounded as well.

Further application for SS-Iterative can be found in any setting where it is desired to solve $A\mathbf{x} = \mathbf{b}$ in a converging and self-stabilizing manner, while A is given, and $\mathbf{b}$ may change slightly from one round to the next. Notice that the analysis given in Section 4 holds in such a system.

As noted in Remark 2 the matrix A is "part of the code". An optional alternative to the current solution is to compute A^{-1} (the inverted matrix of A) beforehand and include it "as part of the code". Thus, each node could locally solve $\mathbf{x} = A^{-1}\mathbf{b}$, and it can be shown that $\mathbf{x}$ will be bounded (as long as $\mathbf{b}$ is bounded). The main problem with such a solution is the connectivity requirements it incurs. In our solution, scalar values are sent in the network only between direct neighbors. The matrix W represents a weighted adjacency graph. Once inverted, the matrix W^{-1} might not be sparse. A non-zero entry $w_{ij}^{-1} \in W^{-1}$ means that node p_i needs to communicate with node p_j. This extra communication might cause the algorithm to lose its self-stabilizing properties, as non-neighboring nodes would require a self-stabilizing overlay network for their communication.

The assumption of a predefined A is suitable for static networks in which the communication graph is predetermined. For dynamic networks, it would be interesting to adjust SS-ITERATIVE to discover the connectivity of the network, inferring the optimal weights dynamically. We assume that after the weights are calculated, the topology of the sensor network remains stable, thus the convergence analysis of Section 4 should hold.

7.1 Relation to Perturbation Theory

A large amount of research focused on the problem of solving $A\mathbf{x} = \mathbf{b}$ when A and $\mathbf{b}$ are not exactly known. That is, let $\hat{A} = A + \delta A$ and $\hat{\mathbf{b}} = \mathbf{b} + \delta\mathbf{b}$, and consider the equation $\hat{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}$; what can be said about $\mathbf{x}$ in relation to $\hat{\mathbf{x}}$?

Our setting is "easier" in one sense, and "harder" in a different sense. In our setting A is known, *i.e.,* $\delta A = 0$. However, $\hat{\mathbf{b}}$ is not well defined. That is, the input vector - which is described by $\hat{\mathbf{b}}$ - changes over time. When solving $\hat{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}$ it is assumed that there is some $\mathbf{b}$ that is *constant* but it was measured with an error. In our case, $\mathbf{b}$ is not constant as it changes over time, while it represents the measurements correctly.

As a future research, it would be interesting to consider the implications of adding inaccuracy to the measurements. The vast body of knowledge regarding perturbation theory would definitely aid in this extension to our model.

7.2 Relation to convex optimization

Many practical optimization problems are given in the quadratic form $f(x) = 1/2\mathbf{x}A\mathbf{x} - \mathbf{b}^T\mathbf{x}$, where the task is to compute $\min_{\mathbf{x}} f(\mathbf{x})$ distributively over a communication network. A survey showing several applications can be found in [3]. Example applications are monitoring, distributed computation of trust and ranking of nodes and data items.

A standard way for solving $\min_{\mathbf{x}} f(\mathbf{x})$ is by computing the derivative and comparing it to zero to get the global optimum. When the matrix A

is symmetric, $f'(\mathbf{x}) = A\mathbf{x} - \mathbf{b} = 0$, and we get a linear system of equations $A\mathbf{x} = \mathbf{b}$. In other words, the convex optimization problem is reduced into a solution of a linear system of equations.

Interior point methods [4, Ch. 11] solve linear programming problems by applying Newton method iteratively. Each computation of the Newton step involves a solution of a linear systems of equations. An area of future work is to examine the applications of our self-stabilizing algorithm to these methods. The difficulties arise from the fact that the matrix A needs to be recomputed between iterations, so nodes need to be synchronized and aware of the current iteration taking place.

Acknowledgements

The authors would like to thank Golan Pundak for assisting with the simulations, and the anonymous reviewers for their helpful comments.

References

1. R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *IEEE International Conference on Data Mining (ICDM'07)*, 2007.
2. D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Calculation. Numerical Methods.* Prentice Hall, 1989.
3. D. Bickson and D. Malkhi. A unifying framework for rating users and data items in peer-to-peer and social networks. In *Peer-to-Peer Networking and Applications (PPNA) Journal, Springer-Verlag*, 2008.
4. Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, March 2004.
5. L. Davidovitch, S. Dolev, and S. Rajsbaum. Stability of multivalued continuous consensus. *SIAM Journal on Computing*, 37(4):1057–1076, 2007.
6. D. Dolev and E. N. Hoch. Ocd: Obsessive consensus disorder (or repetitive consensus). In *Proc. of the 27st Int. Symposium on Principles of Distributed Computing (PODC'08)*, Tornoto, Canada, Aug. 2008.
7. S. Dolev. *Self-Stabilization.* The MIT Press, 2000.
8. S. Dolev and S. Rajsbaum. Stability of long-lived consensus. *J. Comput. Syst. Sci.*, 67(1):26–45, 2003.
9. Jun Fang and Hongbin Li. Distributed event-region detection in wireless sensor networks. In *EURASIP J. Adv. Signal Process*, volume 2008, pages 1–10, New York, NY, United States, January 2008. Hindawi Publishing Corp.
10. Koen Langendoen and Niels Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. In *Comput. Networks*, volume 43, pages 499–518, New York, NY, USA, November 2003. Elsevier North-Holland, Inc.
11. Keith Marzullo. Tolerating failures of continuous-valued sensors. *ACM Trans. Comput. Syst.*, 8(4):284–304, 1990.
12. R. Olfati-Saber. Distributed Kalman Filtering for Sensor Networks. In *Proc. of the 46th IEEE Conference on Decision and Control*, December 2007.