Simulation Study of Average Convergence Rates of Self Healing Distributed Optimization Algorithm

Eric Shang Nan Chen and Randy Freeman

Department of Electrical and Computer Engineering, Northwestern University

I. Introduction

In recent years, the field of networked systems has seen a rapid proliferation of applications ranging from swarm robotics to large-scale machine learning. A networked system refers to a system with a large number of nodes or individual components that contribute toward a common objective [1]. In such systems, it is often unfeasible to compute the global optimal solution for the common objective using a centralized processing element. For a machine learning model that continuously trains itself and makes predictions based on billions of examples collected in real time from users or even physical sensors, it is unrealistic to upload every data point to a central server for several reasons. First, there may be communication bandwidth and computational limitations. Second, relying on a central element limits the resiliency of the network. Third, there are privacy concerns associated with centrally collecting users' personal information [2], [3]. Distributed optimization is the process used to find an optimal solution while circumventing the need for a central processing element. Specifically, a network of n agents minimize a global additive objective function by keeping a local estimate x_i of the global minimizer $x_{opt} = \text{argmin}_\theta \sum_i f_i(\theta)$. The agents reach $x_i = x_{opt}$ by computing the gradients of only their local objective functions and communicating along edges of the graph [3] The present study investigates the performance of the self-healing distributed optimization algorithm proposed in [3] through simulation.

In Ridgley et al. [3], the proposed self-healing distributed optimization algorithm is shown to converge at a linear rate ρ and conform to the same worst-case theoretical lower bound as proven in [4] ($\rho \geq max(\frac{\kappa-1}{\kappa+1}, \sigma)$), where κ is the condition ratio capturing the variation of curvature in the objective function, and σ is a graph connectivity parameter. The primary novelty of the self-healing algorithm is that it guarantees correctness even if there is packet loss, agents drop out, or local cost functions change by not requiring the trajectories to evolve on a specific subspace. The key mathematical difference in [3] compared to previous work is that the location of the Laplacian and integrator are switched, with the integrator feeding into the Laplacian. In the case of packet loss, an additional packet loss protocol that has each node remember the previous states of neighbors and use them for computation in the next iteration is implemented. [3]. A block diagram illustrating the work done by one agent during the algorithm is illustrated in Figure 1, while more detailed pseudo-code and proofs are available in [3].

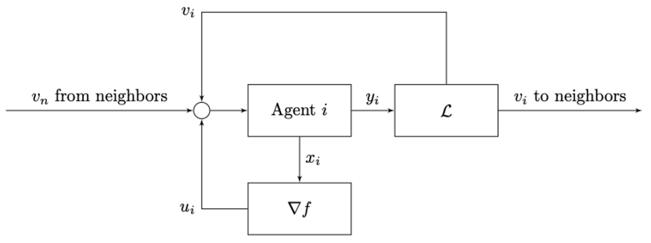

Fig. 1. Block diagram of work done by one agent during an iteration of the algorithm. $\mathcal{L}$ represents the Laplacian block, whereas ∇f represents the gradient of the local cost function.

Although worst-case convergence rates were calculated from the optimization of algorithm parameters, the average convergence rate of the algorithm when applied to real graphs was not explored. Such is the case for much of the literature for distributed optimization algorithms. Nevertheless, the average convergence rate and its improvement over the worst-case is also important to understand for practical applications such as swarm robotics, sensor networks, and distributed machine learning.

A. Research Objectives

The present study aims to bridge the gap between theoretical worst-case convergence rates and realistic ones via simulation on gradient descent for a logistic regression problem. The effect of changing the following independent variables on the convergence rate and convergence time (number of iterations) was explored:

- μ - The average degree of the nodes
- n - The total number of nodes
- σ - A parameter for graph connectivity in the range [0,1] defined by $\sigma = ||I - \Pi_n - \mathcal{L}||$, where $||\cdot||$ denotes the Euclidean norm, Π_n is the projection matrix $\frac{1}{n}\mathbb{1}\mathbb{1}^T$ onto $\mathbb{1}_n$, and $\mathcal{L}$ is the graph Laplacian.
- p - The probability of packet loss
- Graph type - we consider Barabasi-Albert graphs, Erdos-Renyi graphs, and Watts-Strogatz graphs, and Random Geometric Disk graphs.

II. Methods

A. Graph Generation

1) Generating 4 Types of Graphs: To generate the four types of graphs, the Graphs.jl package was used [5]. For each graph type, a function to generate a graph based on a target number of nodes and average degree was created. For Barabasi-Albert, an initial seed graph with $\frac{n*\mu}{2*(n-1)}+1$ nodes is created, and then the remaining nodes are added one by one using the preferential attachment model (each new node attaches to a sample of $\frac{n*\mu}{2*(n-1)}$ nodes with probability proportional to their degree). For Erdos-Renyi, each edge is added with probability $\frac{\mu}{n-1}$. For Watts-Strogatz, a ring lattice is first created, and then each edge is rewired with a probability of 0.01. For Random Geometric Disk graphs, a communication radius is first calculated from finding the root of the function $f(r) = -\mu + (n-1) * \int_0^r P(s)$, where $P(s)$ is the probability function for the distribution of distances s between two points on a disk defined by $P(s) = \frac{4s}{\pi} cos^{-1}(\frac{s}{2}) - \frac{2s^2}{\pi}\sqrt{1-\frac{s^2}{4}}$ [6]. Then, each node is placed randomly on a disk, and an edge is added between two nodes if their distance is less than the communication radius.

2) Generating Graphs with a Target σ: While there are standard methods for generating graphs based on number of nodes and a target average degree, a method for generating graphs based on the σ parameter has not been developed. In this study, we explore two methods. The first involves scaling all of the edge weights by a specific constant, which in turn scales the graph Laplacian and changes sigma. The second method involves changing the graph topology by rewiring the edges and changing the average degree to get a minimum σ value optimized by COSMO.jl [7], a convex optimization solver from Oxford control, that is as close as possible to the target σ value. Datasets of graphs with varying σ values were generated using both methods.

B. Simulation Setup

The main logistic regression dataset used for simulation was the COSMO Chip dataset from [7]. Before running the algorithm, the datapoints are partitioned according to the number of nodes in the graph using a seeded random partition. Then, the following local cost function yielded from the logistic loss function with L2-regularization is used for each node:

$$f_i(x_i) = \sum_{j\in S_i} log(1 + e^{-l_j x_i^T M(d_j)}) + \frac{1}{n}||x_i||^2, \qquad (1)$$

where S_i is the set of datapoints assigned to node i, l_j is the target label of datapoint j, and $M(d_j)$ is the "higher-order polynomial embedding of datapoint j" [3]. From this function, the condition ratio κ is calculated as $\kappa = \frac{nL}{2}$, where L is the Lipschitz constant of the gradient of the local cost function, which can be calculated from the largest eigenvalue of the Hessian.

Using κ and σ, the step size α and three other parameters from [4] were optimized using Brent's method from Optim.jl [8]. To have a reference for calculating the errors at each iteration of the algorithm, the logistic regression problem was first solved centrally using Convex. [9].

To speed up the simulation process, warm-starting with cached parameter values was implemented. For each simulation trial, the algorithm terminated when the maximum error was less than a specified tolerance, or until 40,000 iterations.

C. Calculating Convergence Rates

To calculate the simulation convergence rate, the linear part of the semilog plot of maximum error versus iteration must be found. A sliding window algorithm performing linear fits on sufficiently large intervals was implemented, with the part of the curve with the highest R-squared value taken as the linear part. Then, the convergence rate was calculated as the slope of semilog plot raised to the power of the inverse of the number of iterations in the linear interval.

III. Results

A. Varying Average Degree μ

Through simulating the algorithm on graphs with varying average degree but a fixed number of nodes (40) and σ (0.7), it was found that the convergence rate had no clear relationship with the average degree both in the lossless and lossy case, as shown in Figure 2 and Figure 3.

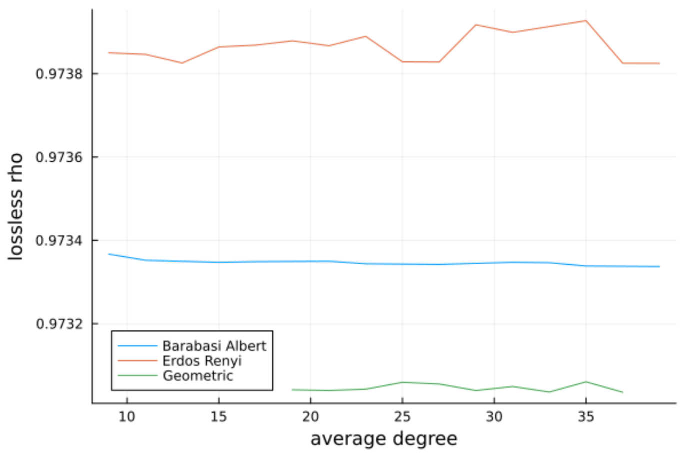

Fig. 2. Convergence rate versus average degree for lossless case.

Some insights regarding the different graph types can also be derived. In the lossless case, there was a clear distinction between the different graph types, with geometric disk graphs having the most desirable convergence rates. However, in the lossy case, the convergence rates of the different graph types were more similar.

B. Varying Number of Nodes n

Similar to varying the average degree, varying the number of nodes with a fixed σ (0.7) did not yield any clear relationship

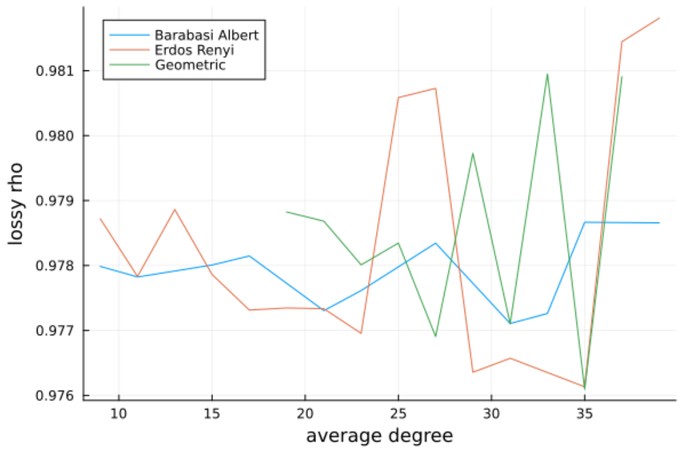

Fig. 3. Convergence rate versus average degree for lossy case (0.3 packet loss probability).

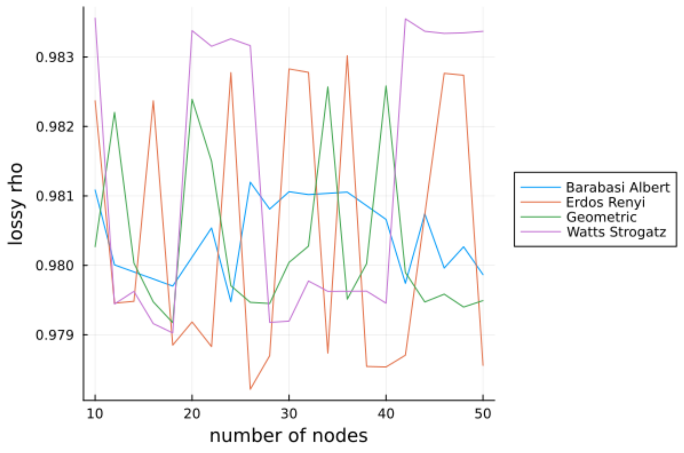

Fig. 5. Convergence rate versus number of nodes for lossy case (0.3 packet loss probability).

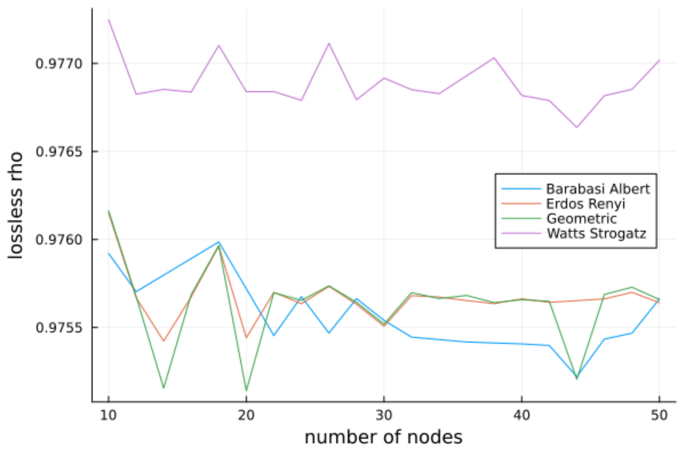

Fig. 4. Convergence rate versus number of nodes for lossless case.

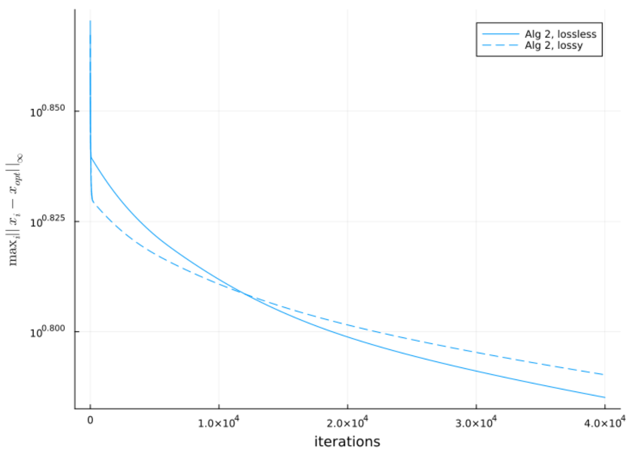

Fig. 6. Semilog plot of maximum error as function of iteration number for large dataset simulation.

between the convergence rate and the number of nodes, as shown in Figure 4 and Figure 5.

To investigate whether the algorithm would perform at a similar level for larger graphs with larger datasets, a logistic regression ”moons” dataset with 10,000 datapoints was generated using the MLJ.jl package [10]. A simulation was carried out on a Barabasi-Albert graph with 300 nodes and $\sigma = 0.85$. The resulting convergence graph, shown in Figure 6, demonstrates that it is moving towards linear convergence, but the error is decreasing at a much slower rate. Due to limitations in computing power, no further exploration of large datasets was done. In future work, more simulation on large datasets to reveal the asymtotic behavior is a potential direction of study.

C. Varying σ

As σ is a key parameter both in defining the theoretical worst-case lower bound for the convergence rate and the calculated worst-case rates through parameter optimization, it was expected that it would have a significant effect on the average convergence rate. Figures 7, 8, and 9 show the lossless and lossy simulation convergence rates as a function of sigma, alongside the optimized worst-case convergence rates. The convergence rates exhibit an approximately linear relationship with sigma. As sigma increases (meaning less connected graph), the algorithm converges slower, which is intuitive. At around $\sigma = 0.6$, there is somewhat of a discontinuity, in which the slope becomes higher after this point. Note that there are spikes in the optimized worst-case convergence rates throughout the curves. This is likely due to errors during the optimization of parameters, and was an issue in the original paper as well.

One key metric of interest for evaluating the performance of the algorithm is the average performance gain over the worst case. To make a simple comparison, the ratio of logarithms of the simulation convergence rate and the worst-case guarantees were calculated. The results are shown in Table I. The average

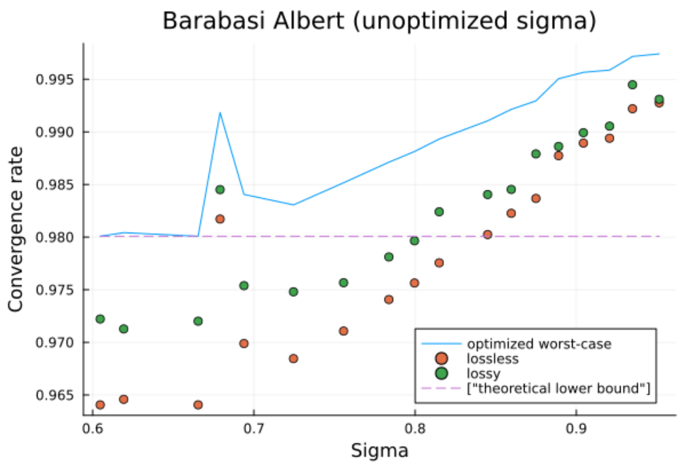

Fig. 7. Convergence rate versus sigma for Barabasi-Albert, n=15.

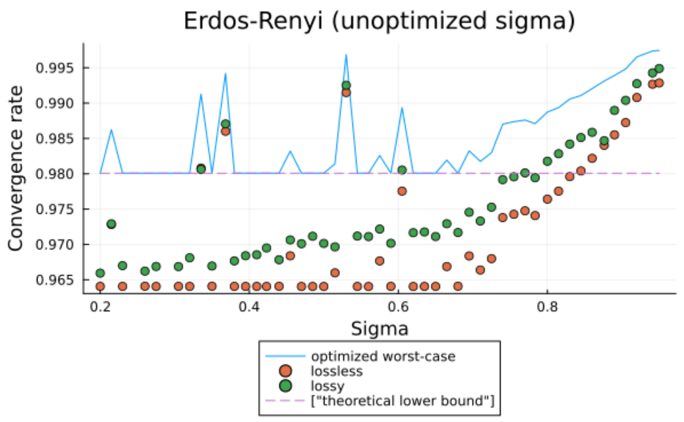

Fig. 8. Convergence rate versus sigma for Erdos Renyi, n=15.

performance gain is around 2.0x for the lossless case and 1.7x for the lossy case. Barabasi-Albert graphs exhibit the highest performance gain.

Type	Lossless log ratio	Lossy log ratio
Erdos Renyi	2.022	1.694
Barabasi Albert	2.204	1.834
Geometric	2.050	1.673

TABLE I
PERFORMANCE GAIN OF SIMULATION CONVERGENCE RATE OVER WORST-CASE CONVERGENCE RATE.

Figure 10 illustrates the logarithmic ratio of the simulation convergence rate and the worst-case convergence rate as a function of σ for Erdos-Renyi graphs. It exhibits a similar trend as in figures 7, 8, and 9, with a sharp nonlinear increase in performance gains at around $\sigma = 0.6$. The plots for Geometric and Barabasi-Albert graphs show a similar trends. The result that the algorithm has higher performance gains for more sparsely connected graphs is not immediately intuitive, but it may be due to the method of calculating the worst-case convergence rate that leads to the estimate being more conservative at higher σ values.

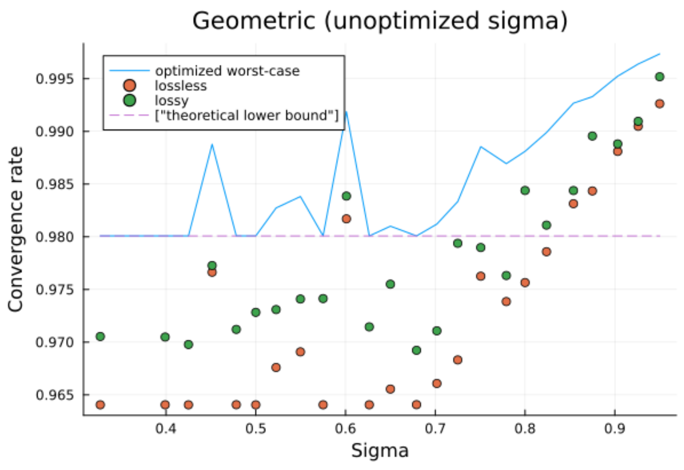

Fig. 9. Convergence rate versus sigma for Geometric Disk, n=15.

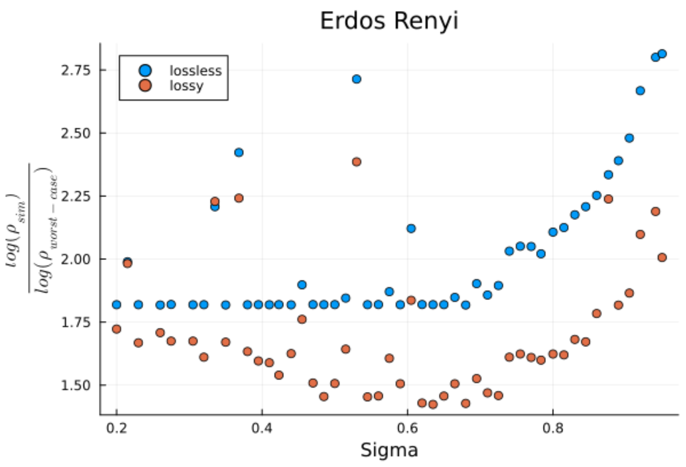

Fig. 10. Logarithmic ratio of simulation convergence rate and worst-case convergence rate versus sigma for Erdos Renyi, n=15.

D. Varying Packet Loss Probability p

Simulation of the packet loss protocol was carried out on a Erdos-Renyi graph with 15 nodes at various packet loss probabilities and four selected σ values. The results are shown in Figure 11. As expected, the convergence rate becomes slower as the packet loss probability increases in an approximately linear relationship. The missing values on the right side of the curves in Figure 11 indicate that the algorithm timed out at 40,000 iterations. The most interesting finding from this graph is that there is a clear inverse relationship between the first packet loss probability at which the algorithm times out and σ, which is counterintuitive because graphs with lower σ are more connected and have overall better performance. Future studies can explore the causes behind this phenomenon further.

Whereas the convergence rate has a linear relationship with packet loss probability, the convergence time (number of iterations) has a nonlinear relationship with packet loss probability.

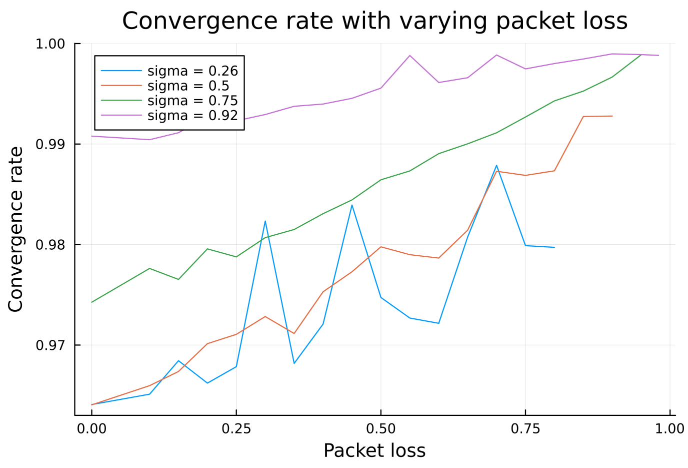

Fig. 11. Convergence rate versus packet loss probability for Erdos-Renyi, n=15.

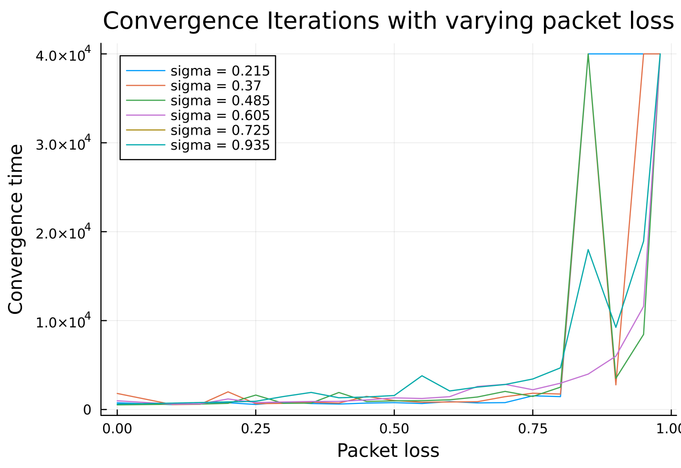

Fig. 13. Convergence time versus packet loss probability for Erdos-Renyi, n=15.

This is shown in Figures 12 and 13. A similar phenomenon where graphs with higher σ reach timeout at lower packet loss probabilities is also observed. In addition, there is a notable difference in the trajectory of the convergence time curves for different σ values. For lower σ values, the convergence time demonstrates characteristics of a phase shift in the interval of 0.8 to 0.9 packet loss probability, meaning the convergence time increases suddenly. For higher sigma values ($\sigma > 0.5$), the trajectory of the convergence time curves resembles a smoother exponential increase.

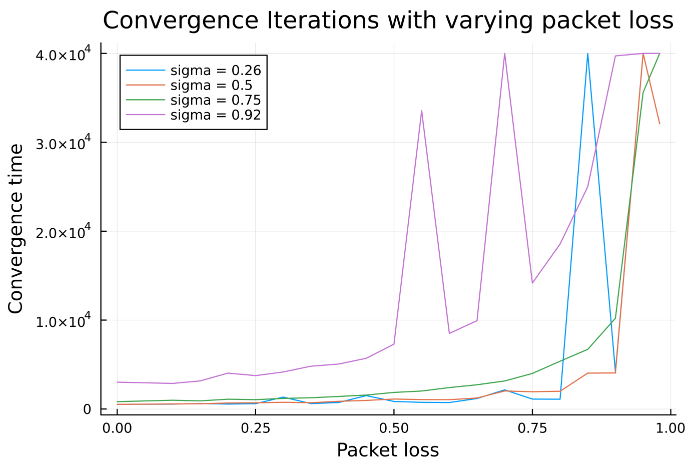

Fig. 12. Convergence time versus packet loss probability for Erdos-Renyi, n=15.

IV. Discussion and Future Work

The results of the simulation study show that the average convergence rate of the self-healing distributed optimization algorithm is not strongly dependent on the average degree or number of nodes in the graph, but is strongly dependent on the σ parameter. The algorithm has a 2x performance gain over the worst-case convergence rate in the lossless case. Furthermore, the packet loss protocol performs well even at packet loss probabilities higher than 0.8, which is a promising result for real-world applications. For example, one swarm robotics application in the Northwestern ECE's Swarm Group has a packet loss probability of around 0.45, which is well within the range of probabilities at which the packet loss protocol converges quickly (within a few hundred iterations). Thus, the algorithm is a promising candidate for distributed optimization tasks in applications such as swarm robotics and sensor networks. For machine learning applications on large datasets, further work may need to be done to compare the algorithm's performance with other algorithms more commonly used in machine learning.

Apart from investigation of the algorithm's performance on large datasets, future work may focus on the underlying reasons for some of the phenomenon identified in this study to further optimize the algorithm, such as the counterintuitive relationship between σ and the first packet loss probability at which the algorithm times out. Moreover, the algorithm should be tested on a wider variety of objective functions beyond logistic regression. Finally, a couple aspects of the algorithm could be considered for improvement. First, the optimization of the parameters, which requires some higher order computations, can be further refined so that errors are minimized, thus eliminating some of the spikes in figures 7, 8, and 9. Second, while the current algorithm is synchronous by iteration, future work can explore asynchronous implementations that may lead to faster convergence.

References

[1] T. Yang, X. Yi, J. Wu, *et al.*, "A survey of distributed optimization," *Annual Reviews in Control*, vol. 47, pp. 278–305, 2019, ISSN: 13675788. DOI: 10.1016/j.arcontrol.2019.05.006. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1367578819300082 (visited on 03/07/2023).

[2] S. Boyd, “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers,” *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2010, ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000016. [Online]. Available: http://www.nowpublishers.com/article/Details/MAL-016 (visited on 03/07/2023).

[3] I. L. D. Ridgley, R. A. Freeman, and K. M. Lynch, “Self-Healing First-Order Distributed Optimization,” version 2, 2021. DOI: 10.48550/ARXIV.2104.01959. [Online]. Available: https://arxiv.org/abs/2104.01959 (visited on 03/07/2023).

[4] A. Sundararajan, B. Van Scoy, and L. Lessard, “Analysis and Design of First-Order Distributed Optimization Algorithms Over Time-Varying Graphs,” *IEEE Transactions on Control of Network Systems*, vol. 7, no. 4, pp. 1597–1608, Dec. 2020, ISSN: 2325-5870, 2372-2533. DOI: 10.1109/TCNS.2020.2988009. [Online]. Available: https://ieeexplore.ieee.org/document/9069430/ (visited on 03/08/2023).

[5] J. Fairbanks, M. Besançon, S. Simon, J. Hoffiman, N. Eubank, and S. Karpinski, *Juliagraphs/graphs.jl: An optimized graphs package for the julia programming language*, 2021. [Online]. Available: https://github.com/JuliaGraphs/Graphs.jl/.

[6] E. W. Weisstein, “Disk line packing,” *MathWorld–A Wolfram Web Resource*, 2023. [Online]. Available: https://mathworld.wolfram.com/DiskLinePacking.html.

[7] M. Garstka, M. Cannon, and P. Goulart, “COSMO: A conic operator splitting method for convex conic problems,” *Journal of Optimization Theory and Applications*, vol. 190, no. 3, pp. 779–810, 2021. DOI: 10.1007/s10957-021-01896-x. [Online]. Available: https://doi.org/10.1007/s10957-021-01896-x.

[8] P. K. Mogensen and A. N. Riseth, “Optim: A mathematical optimization package for Julia,” *Journal of Open Source Software*, vol. 3, no. 24, p. 615, 2018. DOI: 10.21105/joss.00615.

[9] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd, “Convex optimization in Julia,” in *Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages*, IEEE Press, 2014, pp. 18–28.

[10] A. D. Blaom, F. Kiraly, T. Lienart, Y. Simillides, D. Arenas, and S. J. Vollmer, “MLJ: A julia package for composable machine learning,” *Journal of Open Source Software*, vol. 5, no. 55, p. 2704, 2020. DOI: 10.21105/joss.02704. [Online]. Available: https://doi.org/10.21105/joss.02704.