Error Cascades in Collective Behavior

A Case Study of the Gradient Algorithm on 1000 Physical Agents

Melvin Gauci*
Harvard University
Cambridge, MA, USA
mgauci@g.harvard.edu

Monica E. Ortiz*
Harvard University
Cambridge, MA, USA
monica.e.ortiz@gmail.com

Michael Rubenstein
Northwestern University
Evanston, IL, USA
rubenstein@northwestern.edu

Radhika Nagpal
Harvard University
Cambridge, MA, USA
rad@eecs.harvard.edu

ABSTRACT

The gradient, or hop count, algorithm is inspired by natural phenomena such as the morphogen gradients present in multi-cellular development. It has several applications in multi-agent systems and sensor networks, serving as a basis for self-organized coordinate system formation, and finding shortest paths for message passing. It is a simple and well-understood algorithm in theory. However, we show here that in practice, it is highly sensitive to specific rare errors that emerge at larger scales. We implement it on a system of 1000 physical agents (Kilobot robots) that communicate asynchronously via a noisy wireless channel. We observe that spontaneous, short-lasting rare errors made by a single agent (e.g. due to message corruption) propagate spatially and temporally, causing cascades that severely hinder the algorithm's functionality. We develop a mathematical model for temporal error propagation and validate it with experiments on 100 agents. This work shows how multi-agent algorithms that are believed to be simple and robust from theoretical insight may be highly challenging to implement on physical systems. Systematically understanding and quantifying their current limitations is a first step in the direction of improving their robustness for implementation.

CCS Concepts

•**Computing methodologies** → **Multi-agent systems;** •**Hardware** → *Transient errors and upsets;* System-level fault tolerance; •**Networks** → Network performance modeling;

Keywords

Complex systems; spatio-temporal error propagation; multi-agent systems in hardware; reality gap; Kilobots

*Co-first authors.

Appears in: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

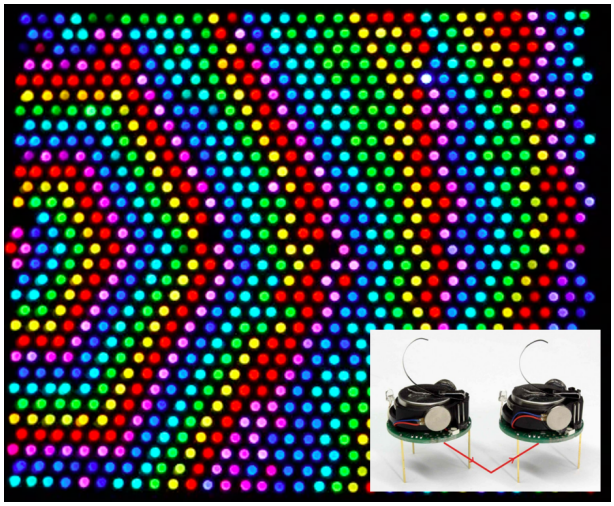

Figure 1: The gradient algorithm running on 1000 physical agents (Kilobot robots, inset). Increasing gradient values are represented by the repeating cycle of colors red, magenta, blue, cyan, green, and yellow.

1. INTRODUCTION

The biological world abounds with self-organizing systems where large numbers of relatively simple agents use local interactions to produce impressive global behaviors, such that the system as a whole is greater than the sum of its parts. These systems exhibit several properties that are highly desirable from an engineering perspective, such as robustness, scalability, and flexibility [3, 4, 12]. Presently, however, there exists a substantial gap between the large body of literature on mathematical and computational models of self-organizing systems and the small body of work on realizable physical systems that could verify the predictions made by such models. These models necessarily involve simplifications, and can fail to manifest emergent behaviors that arise through the intricate physical interactions and variability that exists in real systems. As such, their utility is limited if the fidelity to the physical world remains untested. This leaves a wide gap in our scientific understanding – we critically need physical experimental systems to discover and develop effective mathematical tools for predicting how such complex systems will scale, adapt, or fail.

In this paper we take a well-known collective behavior–the gradient algorithm–and study how it behaves in an physical system as we increase the size of the collective (Figure 1). This algorithm is inspired by biological phenomena such as the morphogen gradients present in multi-cellular development [13]. It is used as a component behavior for many higher-level collective behaviors, such as coordinate system formation [8], message routing and programming in sensor networks [3, 5, 6], and self-assembly in modular robots [3, 10, 11, 12]. Unlike many other collective behaviors, it is well-understood analytically in models of multi-agent systems and has attractive self-stabilizing features [7, see *asynchronous breadth-first search*]. While theoretical models incorporate some non-idealities (e.g. asynchrony, message loss, agent failure/removal), how the algorithm behaves in real large-scale systems has not yet been tested.

We tested the basic gradient algorithm using the Kilobot robot [9], which allows us to create networks of 10 to 1000 agents that communicate wirelessly with their local neighbors. We observed the emergence of *error cascades*, where a single, short-lived agent error is amplified by correct application of the algorithm to cause large scale errors and instabilities in the system. Small-scale experimental systems behave as predicted by analytical models; however as the network size goes up, the system shows a high level of instability, and a 1000-agent network never achieves a stable gradient. We show that these cascades are caused by rare errors not considered before in analytical models, that exploit this algorithm's vulnerability. The rare error is amplified by an agent's neighbors, resulting in an error 'infection' that rapidly spreads both temporally and spatially in a way that is exponentially related to system scale. These error cascades/infections occur in small systems (e.g. 100 agents) but are quickly self-corrected. However, as the system size increases, the entire system exhibits instability due to amplification and overlapping of cascades–a phenomenon which is counter to expectations from previous analytical models.

We present a mathematical model that demonstrates what properties a rare error must have to create a cascade in the gradient algorithm, and how local connectivity amplifies the errors spatially, and local asynchrony amplifies them temporally. Using this model, we are able to predict the probability of a particular agent being in error as a function of both the probability of such rare errors and the network size. We validate our new model with controlled experiments where errors are purposefully injected into a simple line network.

Our experimental case study highlights the need for intermediate systems that can close the gap between existing theory and large-scale implementation by uncovering emergent behaviors and providing a basis for new theory models.

2. AGENT AND GRADIENT MODEL

Consider a network of spatially-distributed agents, in which each agent can communicate with its neighbors within a fixed distance threshold [2, 3, 5, 6]. Within this context, the gradient algorithm assumes the presence of a designated *seed agent* (see Figure 2, right) from which the gradient emanates. The seed agent has a fixed gradient value of zero. Each non-seed agent computes its estimated 'distance' from the seed agent in the form of a hop count: The agent considers its neighbors within a limited visibility range, and computes its own gradient value as the minimum gradient value of its neighbors, plus one. An agent's gradient value represents the minimum number of hops that a message originating from the seed must make in order to reach that agent.

In an ideal scenario, each agent continuously transmits the minimum of its received gradient values from its neighbors, plus one. Under these conditions, the algorithm is provably correct, and converges to a stable equilibrium in linear time with respect to the network's diameter. It is also self-repairing; that is, if there is a change in the network topology (e.g., in the location of the seed), the system re-converges in finite time to correctly reflect this change [7].

In a practical scenario, such as a wireless sensor network, agents communicate with each other by passing discrete messages and do not operate in lock step (i.e., they are asynchronous) [5, 6]. Under these conditions, the gradient algorithm is typically implemented with agents having a polling period during which they collect as many neighbor messages as possible. During each period, an agent collects messages from its neighbors, and transmits its gradient value as computed at the end of the previous period. The period duration is chosen to be large enough so as to minimize the chance of message loss (i.e. of not receiving any message from a neighbor a period), but not too large that it slows down the gradient formation time. If an agent receives no messages (from *any* neighbor) during a period, it retains its gradient value from the previous period. The algorithm as used in this paper is presented below:

```
own value = random number (≥ 1)
min neighbor value = ∞
last updated = timer
while (1) do
    if message received = 1 then
        if neighbor value < min neighbor value then
            min neighbor value = neighbor value
            message received = 0
        end
    end
    if timer > last updated + PERIOD then
        if min neighbor value < ∞ then
            own value = min neighbor value + 1
            min neighbor value = ∞
        end
        last updated = timer
    end
end
```

Algorithm 1: The gradient algorithm as implemented in this work. It is assumed that a "timer" variable is available that increases in the background at a known, fixed rate. It is also assumed that whenever a message from a neighbor arrives, an interrupt handler stores the "neighbor value" and sets the "message received" flag to 1.

3. THE KILOBOT ROBOT

We implemented the gradients algorithm using the Kilobot robot [9] as a physical agent. The Kilobot is a low-cost robot designed for being operated in large collectives, with no need for individual attention. Here we use the Kilobot as a static agent in a wireless sensor network, and will therefore only describe its communication capabilities. Kilobots communicate with each other via an infrared transceiver located at the bottom of the robot, with a communication distance of up to three body lengths. The channel is shared using a carrier sense multiple access with collision detection

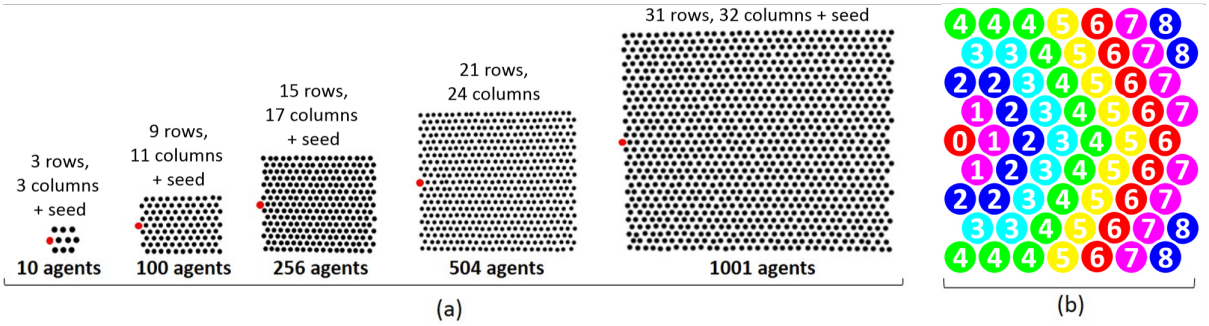

Figure 2: **(a)** Arrangement of agents and seed locations in experiments with several collective sizes; **(b)** Gradient values and display color scheme for an ideal (i.e. error-free) scenario. The red agent with a gradient value of 0 is the seed.

protocol (CSMA/CD), and the user can set the number of transmission attempts per second. Upon receiving a message, a Kilobot can estimate the distance of the transmitting Kilobot based on the signal intensity.

4. EXPERIMENTS ON AGENT NETWORKS

4.1 Experimental Setup and Protocol

We considered networks of increasing sizes, namely 10, 100, 256, 504, 1001 agents. The agents were arranged in a tightly-packed hexagonal configuration, and the neighbors considered in the algorithm were the (up to) 6 first-layer neighbors that are one body length away. This configuration was chosen because it offers a highly controlled experimental setup, with a correct gradient pattern that is regular and predictable (see Figure 2, right)—as such, deviations from this correct pattern are easily recognizable.

For each network size, the agents were arranged in rows containing equal numbers, and the number of rows was chosen to give a roughly square overall configuration. The seed agent was placed roughly in the middle of one of the square's edges. Figure 2 (left) shows these arrangements. For each size of collective, 5 experimental trials were performed. The trial times for 10, 100, 256, 504, and 1001 agents were, respectively, 10, 10, 15, 21, and 30 minutes. These times were chosen to increase roughly in proportion with the diameter of the network, which determines the maximum error-free gradient values that are possible in the system (in this case, 3, 13, 20, 28, and 40). Algorithm 1 was used with a polling period of $T = 2$ seconds and a message rate of around 8 per period. Since Kilobots can receive messages from up to three body lengths away, incoming messages had to be considered or discarded based on the estimated distance of the originating agent. Kilbots have a circular body with diameter 33 mm. Therefore, in a hexagonal configuration, the first layer of neighbors (up to 6) are at a distance of 33 mm. The second layer of neighbors are at a distance of $33\sqrt{3} = 57.16$ mm or $33 \cdot 2 = 66$ mm. In light of these values, a distance threshold of 40 mm was used to distinguish first-layer neighbors from second-layer neighbors and beyond.

The computed gradient values of the agents were tracked using an overhead camera. Each agent has an RGB LED that can in principle display over 16 million colors; however, in order to allow for accurate distinction by the tracking software, the number of colors used was limited to 6. Since the maximum gradient value in the system was larger than 6, each color needed to be re-used for multiple gradient values. As such, the agents displayed the color corresponding to their calculated gradient value, modulo 6. The results 0 through 5 were associated with the following colors, in order: red, magenta, blue, cyan, green, and yellow. An illustration of this color scheme for an ideal (i.e., error-free) scenario is shown in Figure 2 (right). The frame rate of the camera was set to 1 frame per second (i.e., 2 frames per period).

4.2 Results

In order to understand the experimental behavior, we created plots (Figure 3) that allow one to see error patterns linked in time and in space. The horizontal axis shows the time in seconds. *Each row represents the state of an agent over time.* The color represent the agent's gradient value, using the repeating color cycle described in the previous section. The right bar indicates the error-free gradient value of each agent. The vertical axis corresponds to each agent sorted by its error-free gradient value, with the seed being in the top row. Agents with the same gradient value are sorted by ID (which is a function of xy position, but does not guarantee that spatially-adjacent agents are contiguous within a gradient value band).

The plot for 10 agents shows a stable color pattern, which is what is expected from an analytical, error-free model of the gradient algorithm. In a 100-agent collective, errors begin to happen that are correlated in time, often appearing as a downwards color smear across the pattern. This implies that the agents in error believe themselves to be closer to the seed than they actually are. The smears travel in time, with agents closer to the seed being in error first, and agents with higher gradient values being in error at a later time. *Moving downwards, most smears increase in size both spatially and temporally, meaning that more agents are in error for longer periods of time.* These observations are suggestive of errors that propagate and expand in a direction away from the seed. In a 100-agent collective, the gradient values recover between cascades, but in collectives of 256 and 504 agents, the cascades begin to overlap and blend with each other, and by 1000 agents the system is in a constant state of instability, especially far away from the seed.

For each network size, we also computed the proportion of time that each agent spent in error, as a function of the total time. Figure 4 (left) shows these results for each network size, with the agents ordered by increasing (error-free) gradient values. Each line represents an average over all the trials for that network size. It is clear from these plots that proportion of time spent in error increases with the distance from the seed in an exponential fashion. The curves corresponding to 504 and 1001 agents (magenta and cyan, respec-

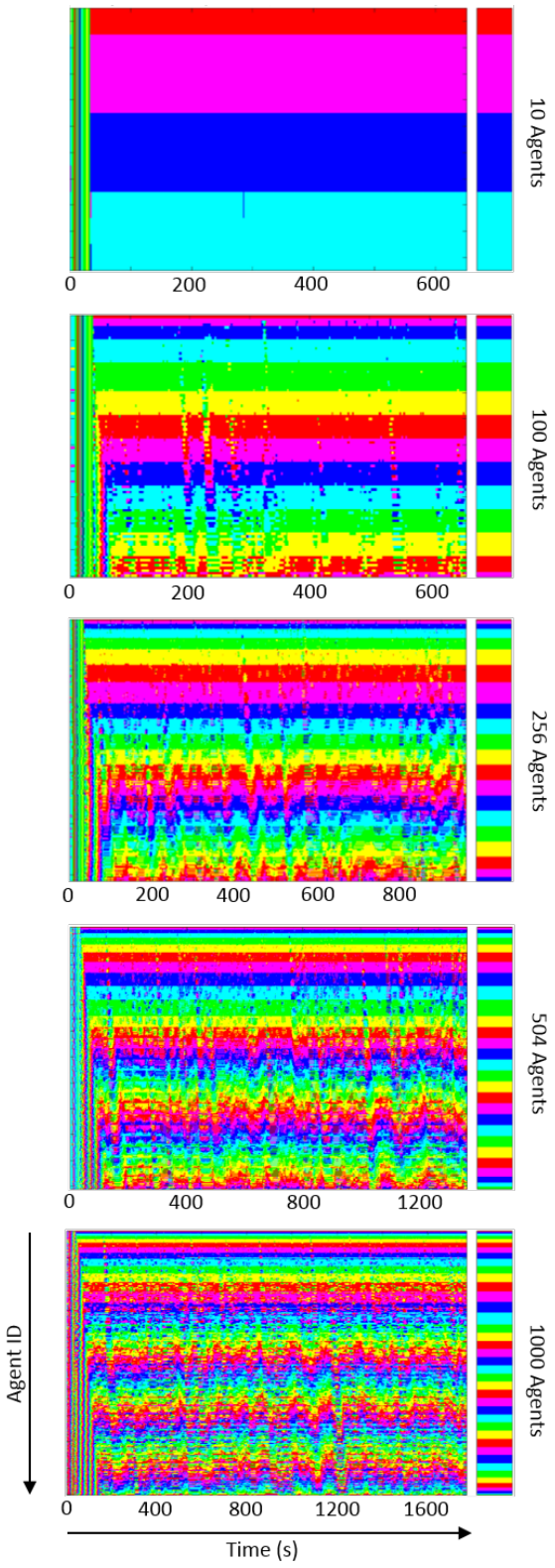

Figure 3: Plots for different sizes of collectives, showing how errors are linked together spatially and temporally. See Section 4.2 for details.

tively) exhibit a seemingly anomalous trend at the highest gradient values, where the proportion of time spent in error *decreases*. However, this is likely to be an artifact and limitation of the experimental setup: Overlapping negative error cascades may bring an agent's gradient value down by 6 units, in which case they would display the same color as if they were not in error, due to the repeating cycle of 6 colors.

Figure 4 (right) shows another perspective of these results for a single trial with a 1000-agent network. The agents are ordered spatially, and the proportion of time in error is indicated by their color. Figure 4 (right) exhibits an interesting 'star-like' pattern. This is a result of the agents' hexagonal configuration, and will be explained in the next section.

4.3 Error Causes and Propagation

Spontaneous errors may be either positive or negative. In the former case, an agent transmits a gradient value higher than its actual one, while in the later case, it transmits a lower value. In this section, we will discuss the potential causes of both types of spontaneous errors, and qualitatively analyze their propagation mechanisms.

4.3.1 Positive Errors are Damped

A common cause of spontaneous positive errors is message loss, the presence of which is expected in most networks. In the gradient algorithm, each agent is anchored to its gradient value by a number of neighbors, which are the ones providing the agent with its minimum neighbor gradient value. If an agent fails to receive a message from *all* of its anchoring neighbors during any one period, then, assuming everything else to be error-free, its minimum neighbor gradient value in that period will be equal to or higher than its own error-free gradient value. Adding one to this value results in a spontaneous positive error by at least one unit, which is transmitted throughout the following period. *Since a spontaneous positive error occurs only if an agent fails to receive a message from* ***all*** *of its neighbors during a period, its occurrence is expected to be relatively unlikely.*

The propagation of positive errors is also unlikely, because the gradient algorithm uses a min() function. Consider the scenario where an agent receives a value that is in positive error from one of its anchoring neighbors in a given period. The effect of this is canceled out if in the same period, the agent receives at least one correct anchoring value from another anchoring neighbor, since this is lower that the erroneous values and will thus be selected by the min() function.

4.3.2 Negative Errors are Amplified

Spontaneous negative errors cannot be caused by message loss; their cases are more subtle and rarer. In a system such as the one we are considering, where agents consider or discard messages based on the distance of the originating agents, one potential cause is noise on the distance sensing. An agent receives a message from another agent with a lower gradient value than its anchoring neighbors, and misjudges the distance of the originating agent to be within the distance threshold. It assumes this value to be its minimum neighbor gradient value, and this results in a lower own gradient value than the true one. A second cause of spontaneous negative errors may be subtle bugs in the algorithm implementation, such as the potential for race conditions due to the presence of interrupt handlers. A third cause of spontaneous negative errors is message corruption. The agents

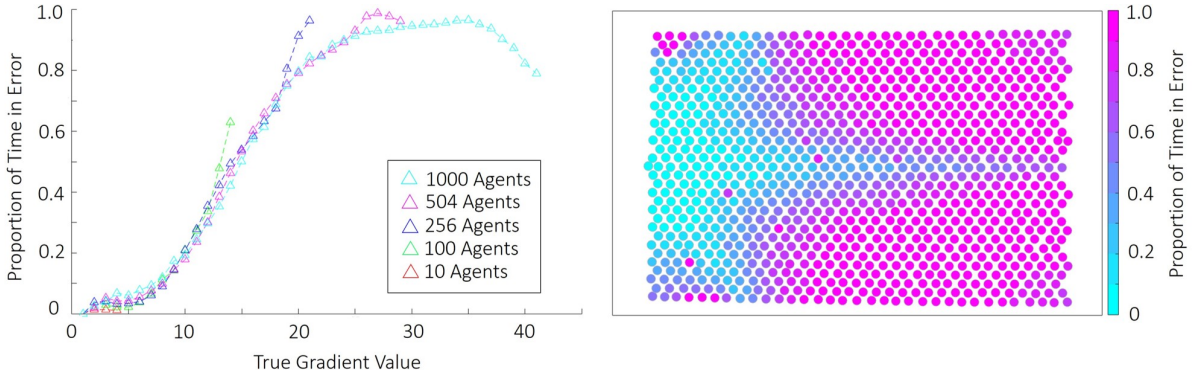

Figure 4: **Left:** Proportion of time spent in error by agents sorted by error-free gradient value, for increasing network sizes. **Right:** Proportion of time spent in error by 1000 agents arranged in a 2-dimensional hexagonal configuration.

typically use a checksum to determine if a received message has been corrupted by the channel; nevertheless, with any error detection/correction code, there is still a non-zero probability of a mistake.

Unlike spontaneous positive errors, spontaneous negative errors are highly contagious. This is due to the min() function used by the algorithm. If an agent receives a message with a lower value than its true anchoring value during some period, this will *definitely* corrupt the subsequent period, and cannot be canceled out by a message with the correct anchoring value (as opposed to the case for positive errors). The 'infection' due to a spontaneous negative error propagates both spatially and temporally.

Spatially, an agent that makes a negative error infects all the neighboring agents that it is anchoring. These, in turn, infect all their neighboring agents that they are anchoring, and so on, leading to a spatial snowball effect (Figure 5, center). As an alternative point of view we can consider the set of agents whose spontaneous negative errors could propagate to a given agent, that is, the region of influence (ROI). Figure 5 (right) shows two example ROIs for a hexagonal configuration of agents. The ROI grows with distance of the agent from the seed. Agents on the axes are only are only affected by the agents in between the seed and themselves; hence, the ROI in this case grows linearly with the distance from the seed. *However, all other agents are affected by a region that grows with the square of the distance from the seed.* This phenomenon explains the results shown in Figure 4.

More surprisingly, our experimental results showed that negative errors also propagate temporally. If the network were synchronous, one period in error by a transmitting agent would cause one period of a receiving agent to be in error. However, asynchrony greatly exacerbates the temporal propagation of negative errors; a phenomenon that has not so far been considered by analytical models. As shown in Figure 5 (right), each period of a transmitting agent in general overlaps with *two* periods of a receiving agent. A single infected period of the transmitting agent may infect either one of these two overlapping periods of the receiving agent, or both. This depends on the phase difference between the periods of the transmitting and the receiving agents, and on how the transmitted infected messages are distributed throughout the period of the transmitting agent. In the worst case scenario (Figure 5 right, top), a single-period spontaneous error by some agent causes the agents that it is anchoring to be in error for two periods, which in turn cause the agents that they are anchoring to be in error for three periods, and so on. In an 'average' case scenario (Figure 5 right, bottom), sometimes one infected period of a transmitting agent only infects one period of a receiving agent. In this case, the number of infected periods grows more slowly with the distance from the originating error than the worst case scenario. Note that a single-period spontaneous error originating from one agent may cause another agent down the line to have *non-consecutive* periods in error (i.e., the infection can 'split'). This is a counterintuitive phenomenon that has not yet been predicted by analytical models.

We have discussed how spontaneous negative errors are caused by rarely-occurring events. In some cases, the rate of occurrence of such events may even be controlled - for example, the probability of accepting a corrupted message can be reduced by introducing more redundancy in the messaging protocol. Owing to their rare nature, these errors have not so far been seriously considered in analytical models. However, as the network size increases, even extremely unlikely events begin to occur frequently. In the next section, we will present a scaling law that shows that for large networks, reducing the *individual* rate of errors has very little effect on reducing the *overall* rate of errors. This, along with the fact that these errors propagate spatially and temporally, means that reducing the individual error rate is not a viable solution for achieving system stability.

5. MODELING ERROR PROPAGATION

The experiments discussed in the previous section demonstrated that negative errors are an Achilles' heel of the gradient algorithm. As such, it is critical to understand precisely their propagation mechanism and quantify their effect on the algorithm's performance. This is a first step towards systematically investigating how the algorithm's resilience to these errors could be improved by making *optimal* tradeoffs (as opposed to ad-hoc modifications).

In this section, we develop an analytical model of the gradient algorithm on 1-dimensional networks that takes into account spontaneous negative errors and their spatial and temporal propagation . In particular, we answer the question: *in a linear network of agents executing the gradient algorithm, given an individual spontaneous error probability, what is the expected proportion of time that each agent spends in error as a function of its distance from the seed?* We validate this model through experiments in which spontaneous negative errors are injected at controlled rates.

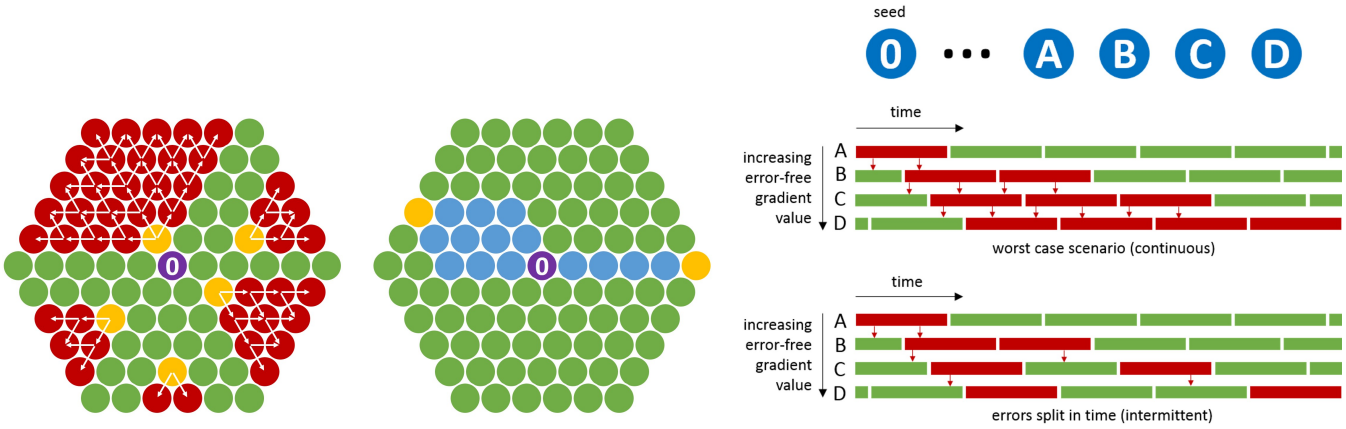

Figure 5: **Left:** Regions of propagation (ROP) for spontaneous negative errors. Yellow agents represent a source of a spontaneous error; red agents represent its ROP. White arrows indicate which agents (may) infect which. The purple agent is the seed. **Center:** Regions of influence (ROI) for spontaneous negative errors. Yellow agents represent 'victim' agents; blue agents represent their ROI. The purple agent is the seed. **Right:** Temporal propagation of spontaneous negative errors.

5.1 Spontaneous Error Model

We consider spontaneous errors that are uncorrelated both spatially and temporally. In other words, the probability of an agent making a spontaneous error in a given period does not depend on (i) whether any other agents are also making errors, and (ii) whether the agent had made any spontaneous errors in the preceding periods. We assume that an agent makes an error in a given period with probability $1/K$, where $K \in \mathbb{R}$ and $K \geq 1$. We can obtain a good approximation for the mean time between spontaneous errors in a collective of N agents by neglecting the asynchrony between the agents' periods, which is a valid approximation as long as K/N is not very small. Since the probability of one agent making a spontaneous error in one period is $1/K$, the probability of at least one agent making a spontaneous error in a one-period window is: $1-(1-1/K)^N$. Therefore, the mean time between spontaneous errors in the collective is:

$$\mathbb{E}(\Delta) = \frac{T}{1-\left(1-\frac{1}{K}\right)^N} \quad \text{seconds} \tag{1}$$

Note that N has a much larger effect on $\mathbb{E}(\Delta)$ than K. For example, in a 1000-agent network with $T = 2$ seconds and $K = 100$, we have $\mathbb{E}(\Delta) = 2$ seconds. Increasing K *tenfold* to 1000 only improves $\mathbb{E}(\Delta)$ to 3.16 seconds.

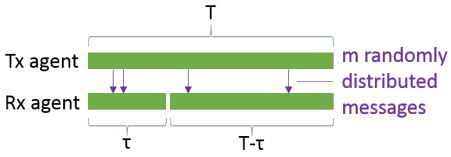

Figure 6: Messaging model approximating CSMA/CD.

5.2 Messaging Model

In general, one period of a transmitting (Tx) agent overlaps with two periods of a receiving (Rx) agent (see Figures 5, right, and 6). Therefore, the gradient value from one period of an agent may reach either one or two periods of a neighboring agent. Let the probabilities of these two events be p_1 and p_2, respectively; naturally $p_1 + p_2 = 1$. We will give three examples of obtaining p_1 and p_2 from a given messaging model.

Example 1: One Message Per Period. Consider the case where during each period, an agent transmits exactly one message. In this case, $p_1 = 1$ and $p_2 = 0$. Note that this is irrespective of when the message is sent out - it could be at a fixed time during the period (e.g., the midpoint), or at a random time with any given distribution.

Example 2: Continuous Transmission. Consider that during each period, n messages are transmitted spaced at regular intervals of T/n seconds. We naturally expect that as $n \to \infty$, $p_1 \to 0$ and $p_2 \to 1$.

Example 3: Randomly-Distributed Messages. We now look at a model that is intended to approximate the CSMA/CD protocol used by the Kilobots (Figure 6). In every period, an agent transmits m messages, whose locations within the period are governed by a uniform random distribution. In general one period of the transmitting agent overlaps with two periods of the receiving agent, as shown in Figure 6. For only one period of the receiving agent to receive any message, the m messages from the transmitting agent must either all fall within the first τ portion of the transmitting agent's period, or all during the final $T-\tau$ portion. Let p_τ and $p_{T-\tau}$ denote the probabilities of these two events occurring, respectively. By observation of Figure 6, we have:

$$p_\tau = \left(\frac{\tau}{T}\right)^m, \quad p_{T-\tau} = \left(\frac{T-\tau}{T}\right)^m, \qquad 0 \leq \tau < T \tag{2}$$

Note that for a specific value of τ, the probability that only one period of the receiving agent is infected is $p_{1,\tau} = p_\tau + p_{T-\tau}$, while the probability that two periods of the receiving agents are infected is $p_{2,\tau} = 1 - p_{1,\tau}$.

However, as the asynchrony between robots is assumed to be unknown, we do not wish for our equations to depend on τ. Therefore, we can obtain an average-case scenario by marginalizing over all possible asynchronies $0 \leq \tau < T$. This gives us the *average case* probabilities that the messages originating from one period a given agent will reach its neighbors in either only one period (p_1), or two periods

(p_2). We have:

$$p_1 = \frac{1}{T}\int_0^T (p_\tau + p_{T-\tau})\, d\tau \tag{3}$$

Evaluating the integral and using $p_2 = 1 - p_1$, we obtain:

$$p_1 = \frac{2}{m+1}, \qquad p_2 = \frac{m-1}{m+1} \tag{4}$$

As a check on Equation 4, we can 'recover' the results from the previous two examples. If $m = 1$, then $p_1 = 1$ and $p_2 = 0$, and we recover the result for one message per period. As $m \to \infty$, $p_1 \to 0$ and $p_2 \to 1$, approaching the result for continuous transmission.

The subsequent analysis relies *only* on p_1 and p_2 as far as the messaging model is concerned. Therefore, p_1 and p_2 are abstracting away the details of the messaging model, and decoupling its details from the rest of the analysis.

5.3 Negative Error Propagation

5.3.1 Time in Error due to One Spontaneous Error

Consider a linear network of agents, and assume that one agent makes a spontaneous negative error for one period. This error will propagate to agents up the line (i.e., away from the seed). We ask the question: *for how many periods, on average, will this spontaneous error cause an agent that is i units up the line to be in error?*. Figure 5 (right, top) shows that the worst case scenario is $i + 1$ periods, which happens when both receiving periods are always infected ($p_1 = 0$, $p_2 = 1$). But if $p_1 \neq 0$, we expect the average case to be less severe than the worst case (see Figure 5 right, bottom). It can be shown analytically that, to a very good approximation with a known error bound, the answer to our question is:

$$\alpha \cdot (i+1) \text{ periods}, \qquad \text{where } \alpha = \frac{(1-p_1)^2}{(1-p_1/2)^2} \tag{5}$$

A full derivation of this result is omitted here due to its length, but is provided in the online supplemental material [1]. The principal idea is to compute the generating function $f(z) = \sum_{i=0}^{\infty} a_n z^n$, where a_n is the expected number of infected nodes i units up the line. This is facilitated by considering the 'width' of the infection at each level i, which is the difference between the first and the last infected periods (irrespective of whether the periods in between are infected), and behaves as a biased random walk.

5.3.2 Time in Error due to All Spontaneous Errors

In a linear network of agents, consider the agent with error-free gradient value g. This has $g - 1$ non-seed agents 'behind' it (i.e., closer to the seed), namely all agents with gradient values $g - i$, $i \in \{1, 2, \ldots, g-1\}$. A spontaneous negative error made by any of these $g - 1$ agents will propagate to agent g, and infect it for some number of periods. Specifically, using the result from the previous section, a spontaneous error by one particular agent $g - i$, will on average infect agent g for $\alpha \cdot (i + 1)$ periods. We ask: *what is the expected proportion of time for which will agent g be in error due to all the spontaneous errors originating from the agents behind it?*

Consider a window of K periods for agent g (where $1/K$ is the spontaneous error probability). In this window, on average, each agent $g - i$ will make one spontaneous error. Without loss of generality, consider one particular period in this K-period window. The probability that this period is *not* infected by a spontaneous error originating from one particular agent $g - i$ is $(K - \alpha(i + 1))/K$. Therefore, the probability that it is not infected by *any* of the agents $g - i$, $i \in \{1, 2, \ldots, g-1\}$, is given by the product:

$$P_g = \prod_{i=1}^{g-1} \frac{K - \alpha \cdot (i+1)}{K} = \left(\frac{K}{\alpha}\right)^{1-g} \prod_{i=1}^{g-1} \frac{K}{\alpha} - (i+1) \tag{6}$$

The probability that some period from the K-period window is not infected (i.e., P_g) approaches the expected proportion of non-infected periods by the law of large numbers. The expected proportion of *infected* periods is therefore $1 - P_g$.

In the rightmost expression of the Equation 6, K and α appear exclusively as a ratio of each other, i.e., K/α. This leads to an interesting observation: Given a spontaneous error probability of $1/K$, and some $0 < \alpha < 1$, this is equivalent to a continuous messaging model (i.e., $p_2 = 1 \implies \alpha = 1$) but with a lower *effective* error probability of 1 in K/α.

Equation 6 is already an expression for P_g; however we can obtain a closed-form expression for P_g if we allow for a slight approximation. In the rightmost expression of Equation 6, note that the product term has the 'flavor' of a factorial function (or more specifically, a binomial coefficient). However, K/α is not in general an integer, and so instead we use the Gamma function, which extends the domain of the factorial function to the real line. Defining $\hat{K} = K/\alpha$, we obtain after some manipulation:

$$P_g = \frac{\hat{K}^{1-g}}{\hat{K}-1} \frac{\Gamma\left(\hat{K}\right)}{\Gamma\left(\hat{K}-g\right)} \approx \hat{K}^{-g} \frac{\Gamma\left(\hat{K}\right)}{\Gamma\left(\hat{K}-g\right)} \tag{7}$$

Finally, we can use Stirling's approximation to the Gamma function, $\ln(\Gamma(x)) \approx \sqrt{2\pi/x}\,(x/e)^x$. Applying this to Equation 7 and rearranging using the laws of indices gives:

$$P_g \approx e^{-g} \left(\frac{\hat{K}}{\hat{K}-g}\right)^{\hat{K}-g-\frac{1}{2}} \tag{8}$$

5.4 Experimental Validation

5.4.1 Experimental Setup and Protocol

We performed an experimental validation of the mathematical analysis discussed above. In particular, we validated Equation 8 with different spontaneous negative error probabilities. 104 Kilobots were used, arranged linearly with the seed located at one end. Each agent was supplied with its true (i.e., error-free) gradient value, which was stored in non-volatile memory. The parameter settings for the gradient algorithm were the same as those described in Section 4.1. The distance threshold of 40 mm now meant that every agent would only listen to messages from its two immediate neighbors (one for the edge agents). At the end of each period, after computing the minimum of its received values + 1, each agent would generate a random number and, with a $1/K$ probability, it would transmit a value of 1 below its computed value for the following period. Otherwise, it would transmit its regular value.

Four sets of experiments were run with different values of K, namely 100, 467, 1508, and 4629. For each value, 5 trials were run lasting 30 minutes each. The agents displayed

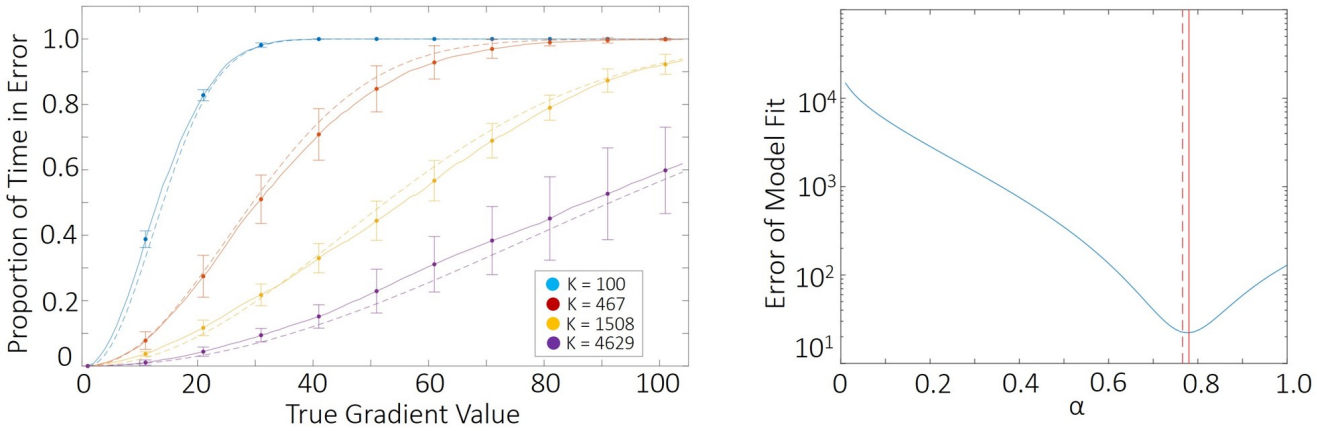

Figure 7: **Left:** Proportion of time spent in error for agents arranged in a linear configuration. Solid and dotted lines represents theoretical predictions and experimental results, respectively. Error bars indicate standard deviations. **Right:** Error of model fit vs parameter α. The solid and dashed red lines correspond to the theoretically-predicted and the experimental minima, respectively.

one of three colors: green if its algorithmic gradient value matched its error-free gradient value; red if its algorithmic hop count was lower than its error-free gradient value, and blue if it was higher. An overhead camera was used to gather this data, and tracking was performed to extract the information.

5.4.2 *Results*

The results are presented in Figure 7 (left). The colors blue, red, yellow and purple correspond to spontaneous error rates of $1/K$ where $K =$, 100, 467, 1508, and 4629, respectively. Solid and dotted lines represents theoretical predictions and experimental results, respectively. Error bars indicate standard deviations. The theoretical results correspond to a value of α that was calculated according to Equation 5 with the probabilities given by Equation 4 (with $m = 8$) (a value of 0.76). Qualitatively we observe a very good fit for all values of K. The standard deviation seems to increase with increasing values of K (i.e., lower spontaneous error probabilities), and for a given value of K, seems start out small at low distances from the seed, then increase, and finally drop again (e.g., for $K = 467$, it increases until around $h = 40$).

In order to test the accuracy of the messaging model we performed a line search over α and for each value computed the error of model fit according to:

$$\alpha^* = \arg\min_{\alpha} \sum_{K \in \mathcal{K}} \sum_{i=1}^{N} |t_{\text{experimental}} - t_{\text{theory}}(\alpha)| \quad (9)$$

where $\mathcal{K} = \{100, 467, 1508, 4629\}$ and $N = 103$ is the number of non-seed agents used in the experiments. The result is shown in Figure 7 (right). The dotted line corresponds to the theoretical value of α, while the solid line corresponds to the optimal value that minimizes the error of fit.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have developed a better understanding of a well-known collective behavior–the gradient algorithm–by testing it on $10 - 1000$ physical agents. The experimental results revealed that this algorithm is highly sensitive to single, short-lived errors, which propagate both spatially and temporally, causing cascades. Despite these errors being extremely rare individually, as the scale of the system increases, they begin to happen with a higher frequency, and prevent the system from stabilizing. Additionally, the experimental results showed that the asynchronous nature of the agents greatly exacerbates the temporal propagation of these errors - a phenomenon that had hitherto not been considered in analytical models of this algorithm.

Given the rapid expansion of error cascades that we demonstrate in this paper, any mitigating solution will require detecting errors near the error source. However, the exact solution will depends on the error model; for instance, isolated errors can be solved by requiring multiple confirmations, but rare bursts of errors may require tracking per-neighbor gradient value stability. Nevertheless, in all cases, the solutions trade off speed of responsiveness to real changes with the ability to prevent cascades from starting—a kind of accuracy-speed trade-off. Current error-mitigating strategies rely on ad-hoc, heuristic rules. By analytically investigating and quantifying multiple error models, we intend to provide in future work systematic, Pareto-optimal solutions for damping error cascades effectively.

The results in this paper highlight the critical need for closing the gap between theory and practice in multi-agent systems, through the use of testbed systems in controlled environments.

Acknowledgments

This research was funded by a DARPA DSO grant and the Wyss Institute for Biologically Inspired Engineering. The authors would like to thank Dylan J. Altschuler for his help with debugging the experimental code and understanding the cause of spontaneous negative errors.

REFERENCES

[1] Authors. Online supplementary material, 2016. `https://goo.gl/rK1uSm`.

[2] J. Beal, J. Bachrach, D. Vickery, and M. Tobenkin. Fast self-healing gradients. In *Proc. 2008 ACM Symp. Applied Computing*, pages 1969–1975, 2008.

[3] J. Beal and R. Schantz. A spatial computing approach

to distributed algorithms. In *45th Asilomar Conf. Signals, Systems, and Computers*, pages 1–5, 2010.

[4] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[5] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proc. 6th Annual Int. Conf. Mobile Computing and Networking*, pages 56–67, 2000.

[6] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Networking)*, 11(1):2–16, 2003.

[7] N. A. Lynch. *Distributed algorithms*. Morgan Kaufmann, San Francisco, CA, USA, 1996.

[8] R. Nagpal, H. Shrobe, and J. Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. In *Information Processing in Sensor Networks*, pages 333–348, 2003.

[9] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *2012 IEEE Int. Conf. Robotics and Automation*, pages 3293–3298, 2012.

[10] M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[11] W.-M. Shen, B. Salemi, and P. Will. Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots. *IEEE Trans. Robot. Autom.*, 18(5):700–712, 2002.

[12] W.-M. Shen, P. Will, A. Galstyan, and C.-M. Chuong. Hormone-inspired self-organization and distributed control of robotic swarms. *Autonomous Robots*, 17(1):93–105, 2004.

[13] L. Wolpert and C. Tickle. *Principles of Development*. Oxford University Press, New York, NY, USA, 4 edition, 2010.