Flexible Self-Healing Gradients

Jacob Beal
BBN Technologies
10 Moulton St.
Cambridge, Massachusetts 02138
jakebeal@bbn.com

ABSTRACT

Self-healing gradients are distributed estimates of the distance from each device in a network to the nearest device designated as a source, and are used in many pervasive computing systems. With previous self-healing gradient algorithms, even the smallest changes in the source or network can produce small estimate changes throughout the network, leading to high communication and energy costs. We observe, however, that in many applications, such as routing and geometric restriction of processes, devices far from the source need only coarse estimates, and that a device need not communicate when its estimate does not change. We have therefore developed **Flex-Gradient**, a new self-healing gradient algorithm with a tunable trade-off between precision and communication cost. When distance is estimated using **Flex-Gradient**, the constraints between neighboring devices are flexible, allowing estimates to vary by an amount proportional to a device's distance to the source. Frequent small changes in the network or source thus cause frequent estimate changes only within a distance proportional to the magnitude of the change, as verified in simulation on a network of 1000 devices. This can enable drastic reductions in the communication and energy cost of gradient-based algorithms.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed programming*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms

Algorithms, Reliability, Experimentation

SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

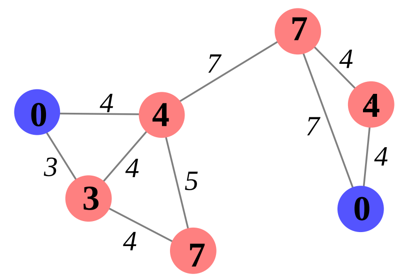


Figure 1: A gradient is a distributed estimate of the distance from each device to a nearest source device (blue). The value of a gradient on a network approximates shortest path distances in the continuous space containing the network.

Keywords

Amorphous computing, spatial computing

1. CONTEXT

A common building block for distributed computing systems is a *gradient*—a biologically inspired operation in which each device estimates its distance to the closest device designated as a source of the gradient (Figure 1). Gradients are commonly used in systems with multi-hop wireless communication, where the network diameter is likely to be high. Applications include data harvesting (e.g. Directed Diffusion[7]), routing (e.g. GLIDER[6]), distributed control (e.g. co-fields[9]) and coordinate system formation (e.g. [2]), to name just a few.

The distance estimates in such systems often must change over time, due to changes in the set of sources, the set of devices, and their distribution through space. Existing self-healing gradient algorithms, such as **CRF-Gradient**[3], adapt their estimates whenever changes occur, so even the smallest changes in the distance estimate may propagate outward through arbitrarily large regions of the network, imposing high communication and energy costs.

We observe, however, that there are many scenarios where, the farther a device is from the source, the coarser the distance estimate that is acceptable. For example, if a pedestrian is carrying a mobile device running a program that spreads to every other device within 50 meters, then it is very important for a nearby device to know whether it is 40 meters or 60 meters away, but not important for a distant device to know whether it is 1040 meters or 1060 meters away. If the pedestrian keeps walking for ten minutes, however, that distant device may become close, and need to know its distance more precisely. Similarly, if two devices are communicating along the shortest path between them, good distance estimates near the path can allow it to shift incrementally, while devices far from the path only become involved during dramatic shifts. Coarseness can be transformed into cost savings by making coarser estimates less likely to change, since devices need not communicate when their estimates are not changing. Thus, for example, a wireless sensor network using gradients to route information from a moving data source to a base station could largely shut down in areas not close to the path between source and base station, saving large amounts of energy.

We have therefore developed **Flex-Gradient**, a new self-healing gradient algorithm with a tunable trade-off between precision and communication cost. When distance is estimated using **Flex-Gradient**, the constraints between neighboring devices are flexible, allowing estimates to vary by an amount proportional to a device's distance to the source. Frequent small changes in the network or source thus cause frequent estimate changes only within a distance proportional to the magnitude of the changes, as verified in simulation on a network of 1000 devices. This can enable drastic reductions in the communication and energy cost of gradient-based algorithms.

2. REVIEW OF SELF-HEALING GRADIENTS

Gradients are generally calculated through iterative application of a triangle inequality constraint. The simplest self-healing estimate of the gradient value $g_x(t)$ of a device x at time t is

$$g_x(t) = \begin{cases} 0 & \text{if } x \in S(t) \\ \min\{g_y(t_{x,y}) + d(x, y, t_{x,y}) | y \in N_x(t)\} & \text{if } x \notin S(t) \end{cases}$$

where $S(t)$ is the set of source devices, $N_x(t)$ is the neighborhood of x (excluding itself), $t_{x,y}$ is the origin time for the information about a neighbor y that is available to x, and $d(x, y, t)$ the estimated distance between neighboring devices x and y. Whenever the set of sources $S(t)$ is non-empty and the network is not changing, repeated fair application of this calculation (sending update messages to neighbors whenever a device's g_x changes) converges to the correct value at every point. We call this limit value, derived from the current configuration of the network and source, $\bar{g}_x(t)$.

Although this naive algorithm converges, it may do so at an arbitrarily slow rate set by the shortest link in the network. The **CRF-Gradient** algorithm[3] addresses this problem by switching between *constraint* and *restoring force* modes (hence the acronym CRF). In constraint mode, the value of a device $g_x(t)$ stays fixed or decreases, set by the triangle inequality from its neighbors' values. In restoring force mode, $g_x(t)$ rises at a fixed velocity v_0. The switch between modes is made with hysteresis, so that the restoring force always overshoots, then snaps down.

Both **CRF-Gradient** and the naive algorithm are very sensitive to changes in $\bar{g}_x$. Any estimate change causes changes that propagate outward to every other device whose estimate depends on the old estimate. Thus even tiny continuing variations in the source or network can cause the entire network to constantly recalculate estimates. This is a problem because, although the estimates may change little, the communication and energy costs of update messages are the same no matter whether the estimate is changing a lot or a little.

2.1 Other Self-Healing Gradients

Self-healing gradient algorithms can be categorized into two general approaches: *incremental repair* and *invalidate and rebuild*. In an incremental repair algorithm like **CRF-Gradient**, devices constantly attempt to move their values up or down towards the correct value. Its predecessors—Clement and Nagpal[5] and Butera[4]'s hop-count gradients and the hybrid distorted algorithm in [1]—also never allow an observable error to persist and are thus sensitive to small changes in the same way as **CRF-Gradient**.

An invalidate and rebuild gradient discards previous values and recalculates sections of network from scratch, generally only allowing values to decrease. Although a scalable proportional response may be possible in this framework, historically these approaches have not taken that route. For example, GRAB[10] allows a single sink, which rebuilds the entire gradient when the error estimate at that point is too high, TTDD[8] builds the gradient on a static subgraph that is rebuilt in case of delivery failure, and TOTA gradients[9] discard and rebuild everything directly dependent on a parent change. Frequent small changes near a critical area in these and related approaches thus lead to repeated recalculation of the entire gradient.

3. NETWORK MODEL

In our design of **Flex-Gradient**, we assume the following network model:

- The network of devices D may contain anywhere from a handful of devices to tens of thousands (or more!).
- Devices are initially distributed arbitrarily through space. If devices are mobile, they move much more slowly than messages propagating through the network.
- Memory and processing power are not limiting resources, though excessive expenditure of either is still bad.
- Execution happens in partially synchronous rounds, once every Δ_t seconds. Each device has a clock that ticks regularly, but frequency may vary slightly and clocks have an arbitrary initial time and phase.
- Devices communicate via unreliable broadcasts to their unit disc neighbors (all other devices within r meters distance). Broadcasts are sent at most once per round, halfway between executions.
- Devices are provided with estimates of the distance to their neighbors, but naming, routing, and global coordinate services are not provided.
- Devices may fail, leave, or join the network at any time, which may change the connectedness of the network.

The **Flex-Gradient** algorithm may not depend on all of these assumptions: they are the constraints on its design.

$$g_x(t) = \begin{cases} 0 & \text{if } x \in S(t) \\ c'_x(t) & \text{else if } max(r, 2c'_x(t)) < g_x(t-\Delta_t), \text{ letting } y \text{ be the min nbr} \\ g_y(t_{x,y}) + (1+\epsilon_x(t)) \cdot \max(\delta \cdot r, d(x,y,t_{x,y})) & \text{else if } s'_x(t) > 1+\epsilon_x(t), \text{ letting } y \text{ be the max-slope nbr} \\ g_y(t_{x,y}) + (1-\epsilon_x(t)) \cdot \max(\delta \cdot r, d(x,y,t_{x,y})) & \text{else if } s'_x(t) < 1-\epsilon_x(t), \text{ letting } y \text{ be the max-slope nbr} \\ g_x(t-\Delta_t) & \text{else} \end{cases}$$

Figure 2: Flex-Gradient algorithm.

4. THE FLEX-GRADIENT ALGORITHM

The sensitivity of previous algorithms appears to be due to their unwillingness to tolerate small errors. We therefore reformulate the goal of a gradient to allow error proportional to the distance from the source, considering $g_x(t)$ to be ϵ-*acceptable* if it is in the range

$$\bar{g}_x(t) \cdot (1-\epsilon) \leq g_x \leq \bar{g}_x(t) \cdot (1+\epsilon)$$

which is identical to the ordinary criteria when $\epsilon = 0$. This leads directly to the basic idea behind **Flex-Gradient**: a device should change its estimate only for significant errors. Since small changes do not produce significant errors except near the source, most estimates need not change in response to small changes, and underlying communication mechanisms on a device can take advantage of this as appropriate to decrease the frequency of broadcasts down toward whatever background level is appropriate to maintain neighborhood relations.

Finding an appropriate distributed test for "significant error" is not simple, however, and many straightforward approaches lead to non-functional algorithms. The solution that we have found is to use the maximum local slope $s_x(t)$ from a device to its neighbors:

$$s_x(t) = \max\{\frac{g_x(t-\Delta_t) - g_y(t_{x,y})}{d(x,y,t_{x,y})} | y \in N_x(t)\}$$

This slope will be 1 when the estimate is exact, and if we wish the estimates to converge to be ϵ-acceptable, then we can simply bound the slope above by $1+\epsilon$ and below by $1-\epsilon$, snapping the estimate to the closest bound when it is too high or too low. Thus, assuming the estimate starts out correct, each hop can absorb up to $\epsilon \cdot r$ in change, propagating only the remnant it cannot absorb onwards.

Changes propagate outward until they are absorbed, so when a region of estimates is already "stretched" or "compressed" to its limit, a small change can propagate outward an arbitrary distance. We therefore need to ensure that devices start with slope 1 and eventually return to it after perturbation. To ensure that devices get correct estimates initially and after non-incremental changes (e.g. appearance of a new source), a device's estimate snaps down to slope 1 when it is more than double a slope 1 estimate through its neighbors and more than one hop from the source, taking estimate through neighbors $c_x(t)$ to be:

$$c_x(t) = \min\{g_y(t_{x,y}) + d(x,y,t_{x,y}) | y \in N_x(t)\}$$

To ensure that devices fix their values eventually to return to slope 1, they compute with $\epsilon = 0$ once every $f \cdot g_x(t)$ seconds, where f is a constant we call the *fixing multiplier*. Thus, devices close to a source tend to correct their estimate more quickly and those farther away will tolerate error longer. Given that $g_x(t)$ may be changing, we calculate the effective error tolerance $\epsilon_x(t)$ using:

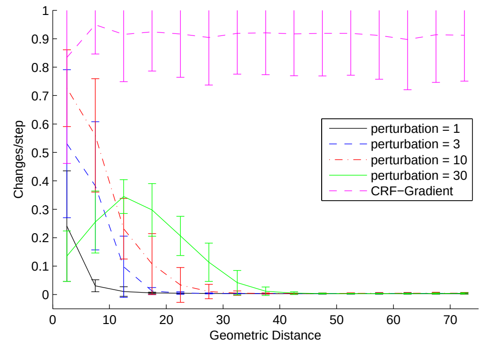


Figure 4: When perturbed by continual changes in the source, Flex-Gradient's estimates vary frequently only within a limited range controlled by the perturbation size, while other self-healing gradients, such as CRF-Gradient, change their estimates throughout the entire network.

$$(\epsilon_x(t), \phi(t)) = \begin{cases} (\epsilon, g_x(t) - \Delta_t) & \text{if } g_x(t) < \phi(t-\Delta_t) \\ (\epsilon, \phi(t-\Delta_t) - \Delta_t) & \text{if } \Delta_t < \phi(t-\Delta_t) \leq g_x(t) \\ (0, g_x(t)) & \text{if } \phi(t-\Delta_t) \leq \Delta_t \end{cases}$$

Thus far, this formulation is subject to the *rising value problem* described in [3]: neighbors mutually constrain one another and because the time between updates is never less than Δ_t, two devices very close to one another can rise only very slowly. The two-mode approach used by **CRF-Gradient** to address this problem does not appear to be usable for **Flex-Gradient**, since it depends on a rapidly spreading loss of constraint. Instead, we introduce a distortion δ into the distance measure, such that neighbor distance is never considered to be less than $\delta \cdot r$. This changes how the slope and estimate through neighbors are calculated:

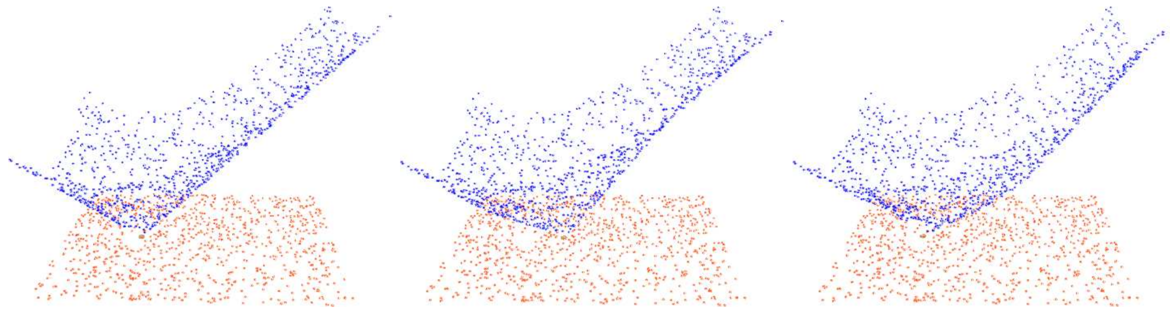
$$s'_x(t) = \max\{\frac{g_y(t_{x,y}) - g_x(t-\Delta_t)}{\max(\delta \cdot r, d(x,y,t_{x,y}))} | y \in N_x(t)\}$$

$$c'_x(t) = \min\{g_y(t_{x,y}) + \max(\delta \cdot r, d(x,y,t_{x,y})) | y \in N_x(t)\}$$

The entire **Flex-Gradient** algorithm can thus be expressed formally as shown in Figure 2.

5. EXPERIMENTS

We verify the effectiveness of **Flex-Gradient** in simulation on a network of 1000 devices distributed randomly through a space 100 meters by 150 meters, with devices



(a) Initial Gradient (b) Source Moving Right (c) Source Returning to Left

Figure 3: Example of Flex-Gradient simulated on a planar network of 1000 devices (red). The network is viewed at an angle, with distance estimates shown as the height of a blue dot above each device. The source (orange) is at the base of the cone of distance values. The slope of the distance estimates is initially uniform (a), but when the source moves to the right (b), then back toward the left again (c), the slope steepens in the direction of motion and becomes shallower in the opposite direction. Repeated motion may leave "wrinkles" in the slope that are straightened out by larger motions or gradual restoration toward unit slope.

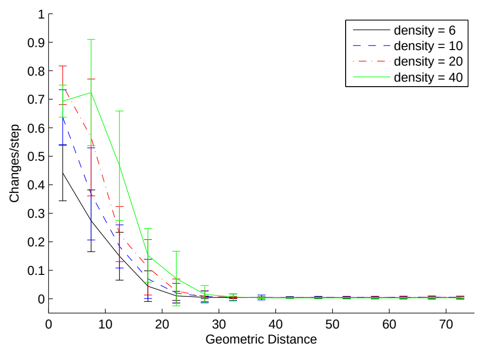


Figure 5: The density of devices in the network does not affect the range in which the perturbation affects estimates, although it does appear to affect the rate at which the estimates change within that area.

executing **Flex-Gradient** steps once per simulated second (Figure 3).

In every trial, the source is a single device that starts in the center of the network, then perturbs the gradient by moving in a circle p meters in diameter at $1/2$ radian per second. The network is allowed to stabilize for 200 seconds (damping out the transient from the initial calculation of gradient values), then data is gathered over the next 1000 seconds of execution. Except where otherwise noted, we set the communication radius r to ensure an expected $\rho = 20$ neighbors per device (making the network about 20 hops in diameter) and use error tolerance $\epsilon = 0.1$, fixing multiplier $f = 10$, distortion $\delta = 0.2$, and a perturbation diameter $p = 10$.

We begin by testing that, under **Flex-Gradient**, continual perturbation of the source causes frequent estimate changes only on devices within a range proportional to the magnitude of the perturbation. We thus ran 20 trials each of **Flex-Gradient** perturbed by a source moving in a circular orbit of diameter $p = 1$, 3, 10, and 30, and of **CRF-Gradient** perturbed with $p = 1$.

Figure 4 shows the relationship between distance to the center of the source's orbit and frequency of estimate changes in populations of devices grouped into 5-meter bands. As expected, **Flex-Gradient**'s estimates only change frequently near the perturbed source, and smaller perturbations lead to smaller affected radii. The observed relation is, surprisingly, even lower than the expected linear relationship, and this may be due to the rapid orbit of the source. With **CRF-Gradient**, on the other hand, even the smallest variation causes near-constant changes in estimates throughout the entire network. Note that for the largest perturbation ($p = 30$), devices near the center of the orbit actually change less than those right on the orbit due to being in the "eye of the storm" of this particular perturbation pattern.

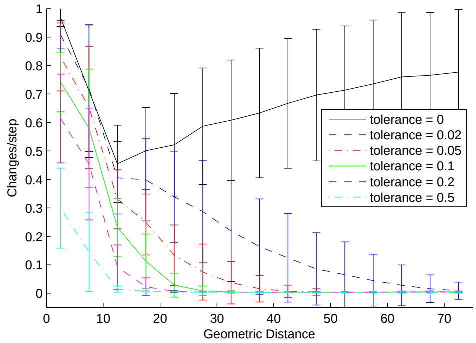
We confirm that this behavior is not affected by the density of devices by running 20 trials each for density $rho = 6$, 10, 20, and 40. Although the rate of change within the affected region does vary, the range in which estimates change frequently is not affected by density (Figure 5).

Finally, we verify that the error tolerance and fixing multiplier affect the algorithm as expected (Figure 6). Running 20 trials each with error tolerance set to $\epsilon = 0$, 0.02, 0.05, 0.1, 0.2, and 0.5 finds that the range affected by the perturbation decreases radically with increased error tolerance, while the quality of estimates is affected only slightly. Running 20 trials each with fixing multiplier set to $f = 1$, 3, 10, 30, and 100 finds that high values allow error to persist longer and low values increase the range at which a perturbation has a small effect, but the choice of parameter does not appear to be particularly sensitive within a wide range.

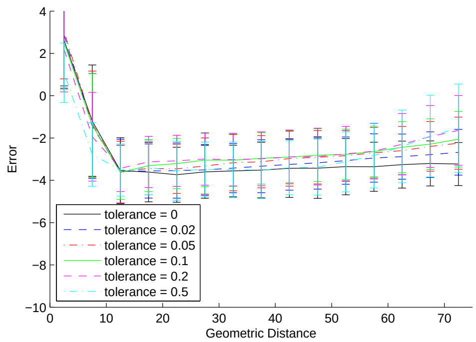
6. CONTRIBUTIONS

We have introduced **Flex-Gradient**, a self-healing gradient algorithm that uses flexible constraints between neighbors to limit the impact of frequent small changes in the network or source. While previous self-healing gradients change distance estimates frequently throughout the network in response to frequent small changes, **Flex-Gradient**'s estimates only change frequently within a distance proportional to the magnitude of the change.

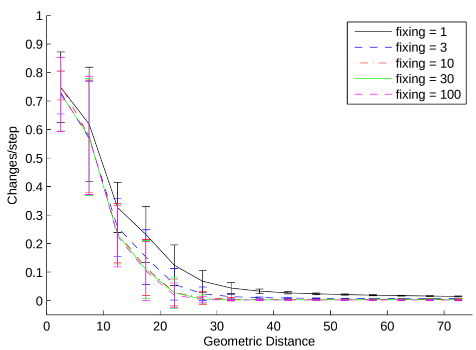
While formal bounds have yet to be established, the experimental results in this paper make it clear that this new algorithm can be expected to greatly reduce the communication and energy costs of gradient-based systems. Thus,



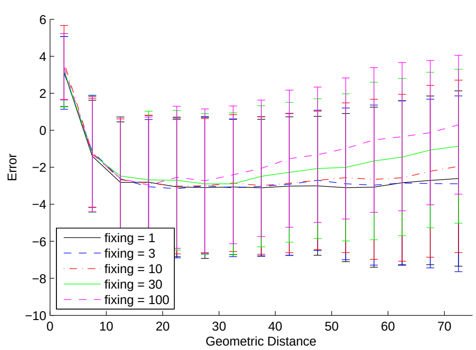
(a) Tolerance: Cost



(b) Tolerance: Error



(c) Fixing: Cost



(d) Fixing: Error

Figure 6: Higher error tolerance decreases the range in which a given perturbation causes estimate changes (a) while affecting estimate quality only slightly (b). Performance appears to be fairly insensitive to choice of fixing multiplier (c, d).

the **Flex-Gradient** algorithm has the potential for significant impact across a wide variety of domains, such as adhoc networking, sensor networks, and swarm robotics, where the combination of volatile networks and sharply limited resources has previously limited the use of gradients and the geometric approach to self-organization that they support.

7. REFERENCES

[1] J. Bachrach and J. Beal. Programming a sensor network as an amorphous medium. In *Distributed Computing in Sensor Systems (DCOSS) 2006 Poster*, June 2006.

[2] J. Bachrach, R. Nagpal, M. Salib, and H. Shrobe. Experimental results and theoretical analysis of a self-organizing global coordinate system for ad hoc sensor networks. *Telecommunications Systems Journal, Special Issue on Wireless System Networks*, 2003.

[3] J. Beal, J. Bachrach, D. Vickery, and M. Tobenkin. Fast self-healing gradients. In *ACM Symposium on Applied Computing*, March 2008.

[4] W. Butera. *Programming a Paintable Computer*. PhD thesis, MIT, 2002.

[5] L. Clement and R. Nagpal. Self-assembly and self-repairing topologies. In *Workshop on Adaptability in Multi-Agent Systems, RoboCup Australian Open*, Jan. 2003.

[6] Q. Fang, J. Gao, L. Guibas, V. de Silva, and L. Zhang. Glider: Gradient landmark-based distributed routing for sensor networks. In *INFOCOM 2005*, March 2005.

[7] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00)*, August 2000.

[8] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang. Ttdd: A two-tier data dissemination model for large-scale wireless sensor networks. *Journal of Mobile Networks and Applications (MONET)*, 2003.

[9] M. Mamei, F. Zambonelli, and L. Leonardi. Co-fields: an adaptive approach for motion coordination. Technical Report 5-2002, University of Modena and Reggio Emilia, 2002.

[10] F. Ye, G. Zhong, S. Lu, and L. Zhang. Gradient broadcast: a robust data delivery protocol for large scale sensor networks. *ACM Wireless Networks (WINET)*, 11(3):285–298, 2005.