Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2007-044 August 24, 2007

Constraint and Restoring Force

Jacob Beal, Jonathan Bachrach, and Mark Tobenkin

Constraint and Restoring Force

Jacob Beal, Jonathan Bachrach, Mark Tobenkin

Abstract

Long-lived sensor network applications must be able to self-repair and adapt to changing demands. We introduce a new approach for doing so: Constraint and Restoring Force. CRF is a physics-inspired framework for computing scalar fields across a sensor network with occasional changes. We illustrate CRF's usefulness by applying it to gradients, a common building block for sensor network systems. The resulting algorithm, CRF-Gradient, determines locally when to self-repair and when to stop and save energy. CRF-Gradient is self-stabilizing, converges in $O(diameter)$ time, and has been verified experimentally in simulation and on a network of Mica2 motes. Finally we show how CRF can be applied to other algorithms as well, such as the calculation of probability fields.

1 Context

Sensor networks are growing both in size and in the complexity of tasks that we want them to perform. As they grow, the challenge of obtaining cheap, responsive, and robust control mechanisms becomes ever more acute. In this paper, we introduce Constraint and Restoring Force, a framework for constructing one such class of control mechanisms. CRF is inspired by the way that local constraints lead to predictable global outcomes in physical systems, and can be used to calculate scalar fields on sensor networks with occasional changes.

To demonstrate the utility of CRF, we apply it to the problem of calculating gradients—measurements of distance to a source region of the network. This is a common building block in sensor network algorithms; applications include data harvesting (e.g. Directed Diffusion[13]), routing (e.g. GLIDER[9]), and coordinate system formation (e.g. [2]), to name just a few.

Applying the CRF framework yields an improved algorithm, CRF-Gradient, which expends little energy when conditions are stable, yet adapts quickly in the face of failures and changing demands. Analysis of CRF-Gradient shows that it is self-stabilizing and converges in time proportional to the diameter of the network. We have implemented the algorithm and verified it experimentally both in simulation and on Mica2 motes.

1.1 Related Work

Our development of CRF depends on previous work in the Amorphous Medium abstraction. This idea was introduced in [3] and developed further in [4]. Recently, we described how the abstraction simplifies engineering of emergent behavior[5] and showed that it can be applied to sensor network problems[1]. While we have preferred to use Proto in our implementation, this algorithm could be implemented, albeit more awkwardly, using other macroprogramming language such as Regiment[15] or Kairos[11].

Physics-inspired approaches are common for distributed control of the positions of mobile agents—see for example the lattice formation in [16], sensor network deployment in [12], and role assignment in [17]. CRF is a more general framework for control of information, with which many of these position-control systems could be implemented.

Adaptable gradients have been previously explored. Most similar to our approach is Active Gradients[7], which increases its distance estimate when communication is disrupted, but assumes knowledge of the maximum range of the gradient and uses only hop-count as a distance measure. Butera's gradients[6] delete and rebuild locally, but losing connection to the source can cause them to thrash.

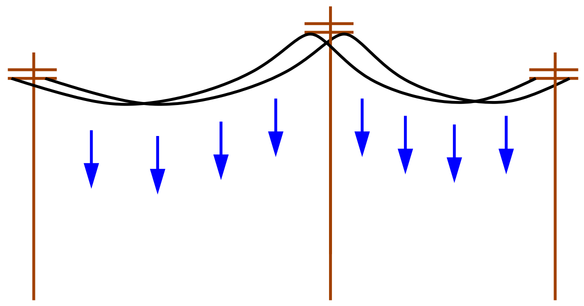
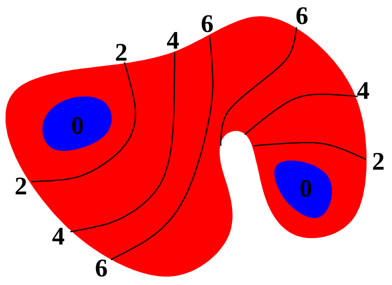
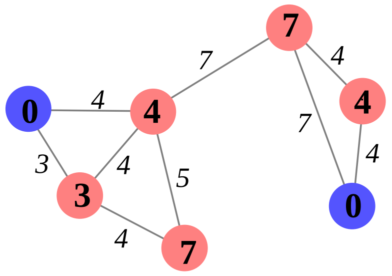


Figure 1: A telephone wire demonstrates CRF: the constraint ensures continuity, gravity is a restoring force pushing the wire down, and the points fixed to the poles determine the overall shape.

More common are approaches which periodically discard and rebuild the gradient, such as GRAB[18] (which uses a single source and rebuilds when its error estimate is too high) and TTDD[14] (which builds the gradient on a static subgraph, which is rebuilt in case of delivery failure). Although these approaches are typically well tuned for a particular use case, the lack of incremental maintenance means that there are conditions that will cause unnecessary rebuilding, persistent incorrectness, or both.



(a) Continuous Gradient



(b) Discrete Gradient

Figure 2: A gradient is a scalar field where the value at each point is the shortest distance to a source region (blue). The value of a gradient on a network approximates the value of the gradient in a continuous space containing the network.

2 Constraint & Restoring Force

We will explain the idea behind CRF with an analogy: think about how telephone wires look. At a few places, they are connected to poles that hold them up in the air. Between the poles, the wire sags downward, pushed evenly by gravity along its length. But there is a limit to how far the telephone wire sags, because two parts of the wire can't be any farther apart than the length of wire between them. These three elements—fixed points at the poles, the wire length constraint, and the force of gravity—interact locally to produce the global shape of the telephone wire.

This is the idea behind CRF: fixed points and constraints define a goal, and the restoring force pushes the system values up against it. Let us make this concrete by applying it to calculation of a gradient. A gradient is a scalar field of numbers across space such that the value at each point is the distance from that point to the nearest point in the source region (Figure 2(a)). When we run a gradient on a sensor network, it will produce a discrete approximation of this calculation across the devices of the network (Figure 2(b)).

Now let's look at this in the CRF framework. Our fixed points are the source region: any point in the source is distance zero. Our constraint is the triangle inequality: the distance from any point to the source is less than or equal to the distance from that point to any neighbor plus the distance from that neighbor to the source. Finally, the restoring force will raise the distance estimate at a constant rate.

Making it work is almost this easy, but not quite. The catch is that information does not move instantaneously, and in order to get the system to converge quickly and without oscillation, points will need to predict possible future values of their neighbors. First though, we will formalize the network model.

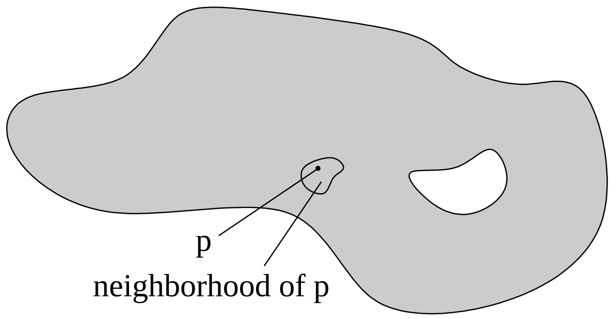


Figure 3: An amorphous medium is a manifold where every point is a computational device that shares state with a neighborhood of other nearby devices.

3 Amorphous Medium & Discrete Network Models

We will develop the CRF framework and algorithms derived from it using the *amorphous medium* abstraction. This abstraction hides many networking details from the programmer, simplifying our algorithm and analysis.

An *amorphous medium* is a theoretical continuous computational material which fills space. The medium is a manifold, where every point is a computational device which independently executes the same code as every other device in the medium.[1]

Nearby devices share state—each device has a *neighborhood* of devices nearby whose state it can access (Figure 3). Information propagates through the medium at a fixed velocity c, so the device accesses values in the past light cone of its neighborhood, rather than the current values (Figure 4).

We cannot construct an amorphous medium but we can approximate one by viewing a network as a discrete sampling of the amorphous medium. Thus, when we develop an amorphous medium algorithm, we first prove it works in the abstraction, then show that the approximate version preserves its functionality.

We will use the following discrete network model:

- The number of devices n is finite and may range from a handful to tens of thousands.

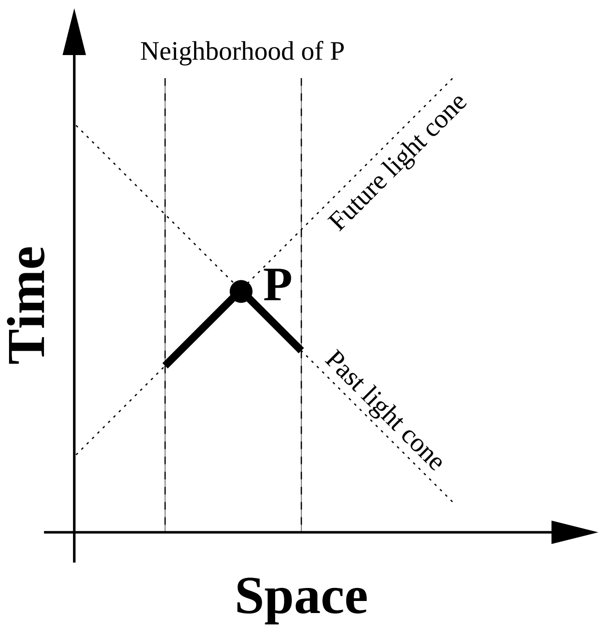


Figure 4: Information propagates through an amorphous medium at a fixed rate, so each processor has access only to values in the intersection of its neighborhood and past light cone.

- Devices are immobile and are distributed arbitrarily through space.
- Devices communicate via unreliable broadcast with all other devices within r units of distance.
- Memory and processing prower are not limiting resources.[2]
- Execution happens in partially synchronous rounds; each device has a clock which ticks regularly, but frequency may vary slightly and clocks have an arbitrary initial time and phase.
- Naming, routing, and coordinate services are not provided.[3]
- Arbitrary stopping failures and joins of points and regions may occur, including changes in the connectedness of the network.

The critical link between the discrete and continuous models is state shared via the neighborhood.

[1]Executions diverge due to randomness, differences in sensor values, and interaction with their neighborhoods.

[2]Excessive expenditure of either is still bad, and memory is an important constraint for the Mica2 implementation.

[3]They may be made available as sensor values, with appropriate characterization of reliability and error.

Each device broadcasts periodically to its neighbors, refreshing their view of its state. The neighborhood of a device is then defined by the set of devices whose broadcasts arrive reliably; when no message from a neighbor has arrived recently, a device assumes that neighbor is dead.

These broadcasts contain intermediate values potentially needed by a neighbor. Because all of the devices are programmed identically, a device can predict what values might be needed by its neighbors and perform any necessary calculations. The broadcasts are used to estimate the space-time displacement of devices relative to one another, using information in the message or its manner of delivery.

The neighborhood implementation is the key to trading off responsiveness and energy efficiency. When neighborhood information is changing, a device broadcasts every round. The broadcasts are sent halfway between computations in order to reduce skew in the rate at which information propagates. When there is no change, the device throttles back its broadcast rate exponentially, multiplying the period between updates by k each time, to a maximum of broadcasting only once every M rounds. Throttle information is included in the broadcast, so neighbors know to expect slow updates and to not prematurely assume the device has died. When the neighborhood information changes again, the device returns its broadcast rate to once per round. Thus, devices broadcast messages at only $1/M$ the normal rate during periods of stability, saving a large percentage of the energy which would otherwise be consumed. For more detail on the discrete implementation see [5].

3.1 Proto

One last excursion before we dive into the algorithms: the code presented below is written in Proto[5], a language we have developed which uses the amorphous medium abstraction. The language is unimportant, but it allows a highly succinct description and can be verified by direct execution on Mica Motes.

We will thus briefly explain enough Proto to allow the reader to understand the algorithm code. Please remember that this paper is about an algorithm and not about the Proto language: for more information on the language or its implementation on Mica2 Motes, see [5] or [1].

Proto is a functional programming language that uses LISP syntax, though it is not a full-featured LISP. An expression in Proto is evaluated globally against an amorphous medium, producing a field (a function assigning a value to each point); an operator takes fields as input and produces a field as output. For example, the expression **(+ 3 5)** takes a field valued **3** at every point and a field valued **5** at every point and adds them point-wise to produce a field valued **8** at every point.

A Proto expression is compiled to execute point-wise on an amorphous medium, where it would evaluate synchronously in regular rounds. When approximated on a network, the rounds are no longer synchronized and neighborhood information is an approximation as described above.

A few Proto operators that we will use are explained below:

- **(inf)** is the numeric value infinity.
- **(fun args ...)** and **(def name args ...)** define anonymous and named functions, respectively.
- **(tup elt ...)**, **(1st tuple)**, and **(2nd tuple)** : **tup** creates tuples, the others access elements of a tuple.
- **(fold-hood fold init value)** is an operation on a device's neighborhood (including the device itself). The operation starts with the value **init** and calls **fold** to merge each **value** in the neighborhood in turn.[4]
- **(nbr-lag)**, **(nbr-range)**, **(radio-range)**, and **(Dt)** are functions that provide space-time information. The **nbr-lag** and **nbr-range** functions can only be used within a neighborhood operation and provide the time and space displacement to the neighbor under consideration. **Dt** gives the elapsed time between rounds. **radio-range** gives an upper bound on possible neighborhood radius.

[4]Accordingly, **value** is included in broadcast messages in the discrete implementation.

```
1  (def set? (x) (< x (inf)))  ; helper: tests whether x is non-infinite
2
3  (def constrain (ct q v rf compare aux)
4    (fold-hood
5      (fun (agg nbr)
6        (let* ((nq (1st nbr)) (naux (2nd nbr)) ; nq=neighbor's value, naux is any other info needed
7               (q_eff (rf nq (if (= v 0) 0 (* 2 (nbr-lag)))))
8               (ncq (ct q aux nq naux))
9               (ncq_eff (ct q aux q_eff naux)))
10         (if (or (= (nbr-range) 0) (compare q ncq_eff))
11             agg
12             (if (not (set? agg)) ncq
13                 (if (compare ncq agg) ncq agg)))))
14     (inf)
15     (tup q aux)))  ; share value and any other info needed
16
17 (def crf (fix init ct rf compare aux)
18   (letfed ((state    ; tuple of (value, rate of change)
19              (tup init 0)  ; initial value
20              (let* ((q (1st state)) (v (2nd state))  ; q=value, v=rate of change
21                     (q_f (fix))
22                     (q_c (constrain ct q v rf compare aux))
23                     (q_r (rf q (Dt))))
24               ;; These are mux, not if, so every point shares values for constraint calculations
25               (mux (set? q_f) (tup q_f 0)
26                    (mux (set? q_c) (tup q_c 0)
27                         (tup q_r (- q_r q)))))))
28     (1st state)))  ; return the value
```

Figure 5: Code for **CRF** and its helper functions.

- **(mux test true false)** and **(if test then else)** are the conditional operators of Proto. **If** runs only the chosen branch, while **mux** runs both, allowing neighbors access to **fold-hood** values from the branch not taken.

- **(letfed ((var init step) ...) ...)** establishes state via a feedback loop. Starting at **init**, each round **var** is calculated by evaluating **step** with its previous value.

4 Algorithm

We can now move on to explaining the CRF algorithm and how we apply it to calculate gradients and other functions. Remember, the algorithm is not dependent on the language; if you are uncomfortable with LISP-like languages, please consider the listings as pseudocode.

4.1 CRF

The Constraint and Restoring Force algorithm (Figure 5) is parameterized by the three CRF elements plus an initial value **init**, as shown on line 18:

- Fixed values are supplied by the function **(fix)**, which returns either the fixed value or infinity for unfixed points.

- The constraint has three parts: **ct** is a function that calculates the constraint between neighbors given their values and supplementary information **aux**. Constraint values are then compared with **compare** (usually $>$ or $<$) to find the most dominant.

- Restoring force is applied with the function **(rf q Dt)**, which takes a value **q** and evolves it without constraint over time **Dt**.

CRF works by tracking two state variables: the value of interest **q** and its rate of change **v** (line 19). Every round, a point considers the constraint from its neighbors (line 23). If the point has a fixed value, **q** is set to the fixed value by an impulse, leaving the system with zero velocity (line 26). Otherwise, it moves with an impulse if constrained (line 27), or applies the restoring force if not (line 28). Finally, **CRF** returns its current value.

In the function applying the constraint (line 3), each point shares its value and an optional user value

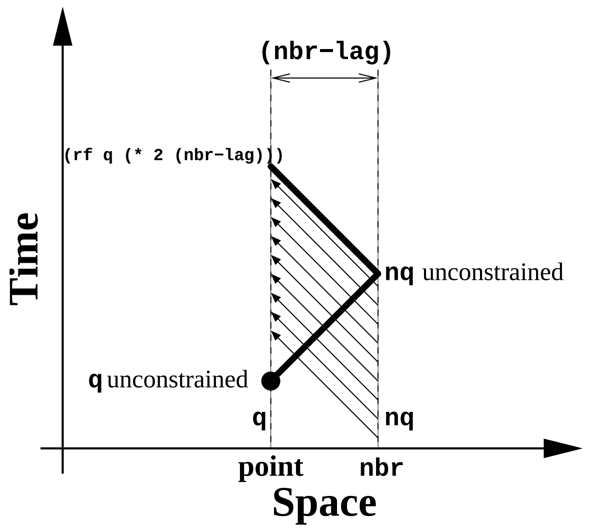


Figure 6: When a point becomes unconstrained, the direction of possible new constraints may differ from the direction in which the wave of deconstraint passed through it. When it considers a potential constraint from a neighbor, then, it must project the unconstrained value by twice the lag between them.

(line 16) and defaults to being unconstrained (line 15). A point compares its value with the constraints imposed by its neighbors, not including itself (line 10). Because communication is not instantaneous, however, this comparison must compensate for time lag regarding possible changes in the direction of constraint.

When a point becomes unconstrained, it can become constrained again through any neighbor, in any direction. The change from constrained to unconstrained, however, is a wave flowing through the neighborhood from the direction of the old constraint. Thus when an unconstrained point compares its value to that of a neighbor, the possible new constraint vector may differ by twice the lag between them (Figure 6). Accordingly, an unconstrained point must compensate by projecting the unconstrained future value of its neighbors (line 7). Ignoring neighbors whose projected values do not constrain it (line 11), the point takes the neighbor value with the tightest constraint (line 13, 14).

If the restoring force acts slowly enough compared to the speed c at which information propagates, the algorithm will be stable even without compensation. In the discrete approximation, however, time and space are discretized independently, and this can cause c to be very slow over short distances.

4.2 CRF-Gradient

Now that we have the CRF framework, using it to build gradients is easy:

```
(def triangle−inequality (q aux nq naux)
  (+ nq (nbr−range)))

(def crf−gradient (src)
  (crf (fun () (mux src 0 (inf)))
       0
       triangle−inequality
       (fun (q dt)
         (+ q (* dt (/ (radio−range) 8))))
       < 0))
```

The fixed values are zero at the source, and the initial value at every point is zero. The constraint is the triangle inequality, which is a minimizing constraint, so we set **compare** to <. The triangle inequality needs no information besides the value and geometry, so we set **aux** to an arbitrary constant 0. Finally, we set the restoring force to raise the value at a small constant rate.

4.3 CRF-Max-Probability

CRF can be applied to other algorithms as well. For example, CRF can be used to calculate maximum cumulative probability paths to a destination—a component used in building a threat avoidance system such as [8]. The maximum probability calculation is:

```
(def line−integral (q f nq nf)
  (* nq (pow (/ (− 2 (+ f nf)) 2)
             (* 0.01 (+ 1 (nbr−range))))))

(def crf−max−probability (dst f)
  (crf (fun () (mux dst 1 (inf)))
       0
       line−integral
       (fun (q dt) (* q (pow 0.99 dt)))
       > f))
```

This algorithm uses a maximizing constraint based on the approximate line integral between two points, so **compare** is set to >. Since we are integrating the field function **f**, we pass it to CRF as **aux** so that it can be accessible to the constraint function. The restoring force operates in the opposite direction, shrinking the calculated probability exponentially. Finally, the destination has a fixed probability of 1, and every point starts with zero probability, reflecting the lack of a known path to the destination.

5 Analysis of CRF-Gradient

We will show that **CRF-Gradient** converges in $O(diameter)$ time by proving self-stabilization, where the network converges to correct behavior from an arbitrary starting state.

Our strategy will be to first prove self-stabilization for an amorphous medium, then adapt the proof for approximation on the discrete network.

For the purposes of this proof, we will assume that the source region remains fixed, and that there are no failures. We will additionally use the following definitions and assumptions:

- A is the amorphous medium manifold. A is smooth, compact, Riemannian, and has finite diameter and dimension. $S \subseteq A$ is the source region. We will consider the source region to be constrained.
- There is an $r > 0$ such that at every point x a closed ball of radius r centered at x is entirely contained within the neighborhood of x.
- $q_{R,t}$, is the set of values in a region $R \subseteq X$ at time t and consists of real numbers in the range $[0, \infty)$.
- c is the constant velocity of information propagation in the medium, f is the constant velocity of q imparted by the restoring force, and $f \leq c/4$.
- Given two points x and y, $d_{x,y}$ is the distance between them and $L_{x,y}$ is the communication lag between them. The distance and lag between regions will be the greatest lower bounds, $d_{X,Y} = \inf(\{d_{x,y} | x \in X, y \in Y\}$ and $L_{X,Y} = \inf(\{L_{x,y} | x \in X, y \in Y\}$. Given a distance d, L_d is the lag across distance d.
- Evaluation is continuous (i.e. rounds happen infinitely often).
- If the source region is non-empty, correct behavior is for every point x to assume a value $q_x = d_{x,S}$ equal to the shortest path to a point in the source region. If there is no source, correct behavior is for the value at every point to rise uniformly at rate f.

Given these definitions, we can begin the proofs, starting with a simple bound showing how the least values in a region control the whole region.

Lemma 5.1. *Let $R \subseteq A$. At time $t = 0$, let $q_0 = \inf(q_{X,0})$, and define the minimum region M as the limit of $M_\epsilon = \{closure(x) | q_{x,0} < q_0 + \epsilon\}$ as $\epsilon \to 0$. Then at time t, every point x with $d_{x,M} < ct$ has value $q_{x,t} < q_0 + ct + 2tf$.*

Proof. Because A is bounded, metric, and compact, M is non-empty.[5] At time L_r, every point y within $d_{y,M} \leq r$ units of the minimum is constrained to $q_{y,L_r} \leq q_0 + r + 2L_r f$. Chaining this outward, we know that x is constrained to $q_{x,L_{x,M}} \leq q_0 + d_{x,M} + 2L_{x,M} f$. If x is never constrained thereafter, it will continue to rise for a period of $t - L_{x,M}$, to a maximum value of $q_{x,t} \leq q_0 + d_{x,M} + 2tf$. Since $d_{x,M} < ct$, we thus have $q_{x,t} < q_0 + ct + 2tf$. □

We also need the converse, a condition under which a region will have no constraints.

Lemma 5.2 (Floating Island Lemma). *Let $R \subseteq A$ be a region where the maximum difference between any two points x and y is $q_{x,t} - q_{y,t} < d_{x,y} + L_{x,y} f$,[6] and all points are unconstrained. Then, unless R is acted on by a constraint from outside, no point within R will be constrained in the future. We will call a region of this type a "floating island."*

Proof. Assume x and y are neighbors, with $q_{x,t} > q_{y,t}$. Then x is receiving the value y held at time $t' = t - L_{x,y}$, which is bounded $q_{y,t'} \geq q_{y,t} - L_{x,y} f$. If x is to remain unconstrained, it must be the case that $q_{x,t} < q_{y,t'} + d_{x,y} + 2L_{x,y} f \leq (q_{y,t} - L_{x,y} f) + d_{x,y} + 2L_{x,y} f = q_{y,t} + d_{x,y} + L_{x,y} f$, which is satisfied by assumption. Since this holds for any pair of neighbors, we can chain neighborhoods together to show that it holds for any pair of devices. □

[5]Even if there is no point with value q_0, we can use an occupancy argument to construct a Cauchy sequence converging to some point in M.

[6]This is effectively a bound on the derivative of q_R, but is formulated as a difference for easier translation to the discrete network.

We can now show self-stabilization in the case where there are no sources.

Theorem 5.3 (Floating Island Growth). *Let $R \subseteq A - S$ be a region without sources. Take the initial time to be $t = 0$ and identify the minimum q_0 and minimum region M as before. Then at time t, the set $\{x | d_{x,M} < ct/2\}$ is a floating island unless acted on by a constraint from outside.*

Proof. Assume for contradiction that this is not true. Then either there is a point $x \in R$ which is constrained or there is a pair of points $x, y \in R$ such that $q_{x,t} - q_{y,t} \geq d_{x,y} + L_{x,y} f$.

Case 1: assume there is a constrained point x. Since $d_{x,M} < ct/2$, then by Lemma 5.1 its value at t is bounded $q_{x,t} < q_0 + ct/2 + 2tf = q_0 + ct$. Since it is constrained, x has a neighbor y such that $q_{y,(t-L_{x,y})} < q_0 + ct - d_{x,y}$. This point y must in turn have constrained in an interval with $t - L_{x,y}$ contained within its limit. This gives a sequence of dependencies at times converging to $t_0 = 0$. Since distance drops at least as fast as time, however, the limit of this sequence is a point l with value $q_{l,0} < q_0$, giving us a contradiction.

Case 2: assume there are two points, x, y such that $q_{x,t} - q_{y,t} \geq d_x y + L_{xy} f$. Then, by the intermediate value theorem, for any ϵ in the range $0 < \epsilon < r$ there must be two points x^ϵ, y^ϵ which also violate the bound and are neighbors with distance $d_{x^\epsilon,y^\epsilon} < \epsilon$. For this to hold, however, y^ϵ must have been constrained by another point z in the interval $[t - L_\epsilon, t]$, or else x would have already been constrained by y, shrinking the difference to within the bound. The value of z is bounded $q_{z,(t-L_{y,z})} \leq q_{y,t} - d_{y,z}$, and it too must have been recently constrained. We can thus construct a sequence of dependencies as in Case 1, halving ϵ at each step, to produce a limit point with a value less than the minimum, again yielding a contradiction. □

Corollary 5.4. *If the source region S is empty, then within $2 * diameter/c$ time every point will be part of a floating island.*

Finally, we can use floating islands to show self-stabilization for an amorphous medium with sources.

Theorem 5.5. *The* **CRF-Gradient** *algorithm self-stabilizes in $4 * diameter/c$ time on an amorphous medium.*

Proof. First, note that once a point is constrained by the source, it will always be constrained by the source—it can only relax towards a shorter path. The relaxation is finished within the transit time of information along the shortest path to the source.

Repeated time-shifted applications of Lemma 5.3 and Lemma 5.2 show that the minimum in the area not constrained by the source rises at rate f.

If the source region is empty, then Corollary 5.4 gives self-stabilization in $2 * diameter/c$ time.

If the source region is not empty, then at time t, the region $\{x | d_{x,S} < ct/4\}$ is constrained by the source, since for any ϵ in $0 < \epsilon < r$, a point y within ϵ of a source-constrained region will become constrained by the source in $2L_\epsilon$ time after reaching value $d_{y,S}$. Since $f \leq c/4$, the rise rate is the limiting factor. The worst-case rise is from minimum value zero, so a point y at distance $d_{y,S}$ becomes constrained at a time $t \leq d_{y,S}/f = 4d_{y,S}/c$. Thus, the entire network is constrained by the source within $4 * diameter/c$ time. □

Theorem 5.6. *The* **CRF-Gradient** *algorithm self-stabilizes in $6 * diameter/c$ time on a discrete network with no collinear neighbors.*[7]

Proof. On a discrete network, time and distance are quantized separately, meaning that $L_{x,y}$ is no longer proportional to $d_{x,y}$ within a neighborhood. Normalizing time to rounds, $L_{d \leq r}$ is in the range $[1/2, 3/2]$: it will be approximated in algorithmic use as the maximum, and c will be approximated as $\frac{2}{3}r$ per round, with r equal to the broadcast radius.

Non-collinear neighbors means that if a device x is constrained by a neighbor y, then no device constraining y is a neighbor of x. With this guarantee, we can bound sequences, given a maximum of two steps to traverse distance r; without, we cannot bound them, because the potential lag is multiplied by the degree of collinearity.

The amorphous medium proofs are affected as follows:

[7]Such a network can be produced by adding a small amount of randomness to geometric information.

- **Lemma 5.1:** The minimum region M is still guaranteed to be non-empty. The lag is no longer constant, so the constraint of a point x may happen earlier, giving more time for it to rise, up to $2/3t$. This shifts the bound to $q_{x,t} < q_0 + ct + \frac{8}{3}tf$

- **Lemma 5.2:** Unchanged, since constraint application uses the maximum value of $L_{x,y}$.

- **Lemma 5.3:** Floating islands grow more slowly, encompassing only everything within $ct/6$ of M: the dependency sequences stretch further in time, with distance dropping as slowly as 1/2r every 3/2 rounds, which is 1/2c. The initial value can also be higher due to the differences in Lemma 5.1, applying an additional 2/3 multiplier to the guaranteed distance. All told, instead of everything within $ct/2$ being part of a floating island, it is reduced to everything within $ct/6$. Time-shifted application still provides a minimum that rises at the same rate.

- **Corollary 5.4:** The time is raised proportionally to $6 * diameter/c$ for everything to be included in a floating island.

- **Theorem 5.5:** The time to become constrained by a source-constrained neighbor converges to $2L_r$ instead of zero, adding that constant to the convergence rate. The critical path, however, switches to the empty source case, with convergence time $6 * diameter/c$. The expected case for random distributions, however, is still $4 * diameter/c$.

$\square$

6 Experiments

We have verified the CRF framework and derived algorithms both in simulation and on a network of Mica2 Motes.

6.1 Simulation

In simulation, **CRF-Gradient** converges and reconfigures as predicted by our analysis. For example, the reconfiguration shown in Figure 7 takes place on a network of 1000 devices, distributed uniformly randomly to produce a network 19 hops wide. Analysis predicts that the reconfiguration should complete within $6 * diameter/c = 6 * 19 = 114$ rounds, and in fact it completes in 74 rounds.

Likewise, in simulation **CRF-Max-Probability**, used in a threat avoidance program on the same network, allows the program to reconfigure proportional to the longest path of change, as shown in Figure 8.

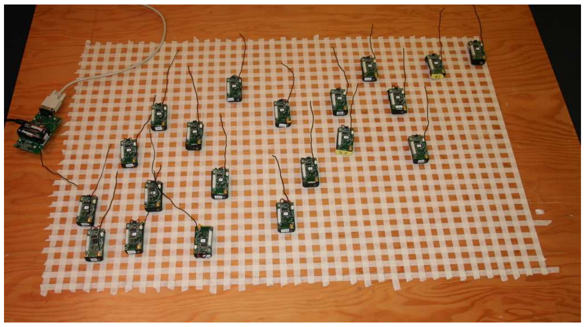


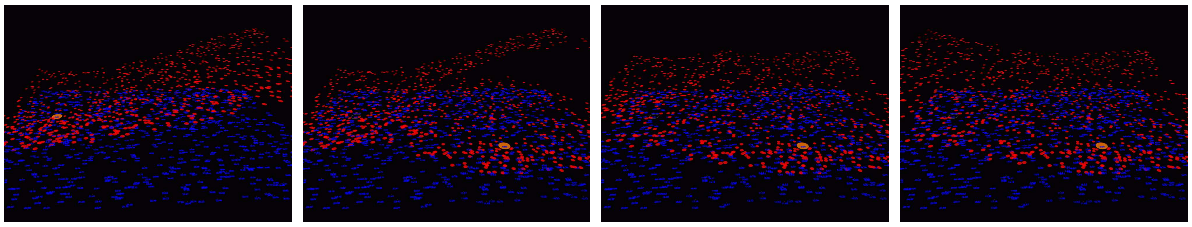
Figure 9: Our experimental network of 20 motes, laid out in a mesh-like network with synthetic coordinates. Reception range was software-limited to 9 inches, producing a 7-hop network. Data is gathered via the monitor device at the left.

6.2 Mica2 Motes

We tested **CRF-Gradient** on a network of 20 Mica2 Motes running Proto on top of TinyOS[10]. The motes were laid out at known positions in a mesh-like network and supplied with perfect synthetic coordinates (Figure 9). Note that, due to the self-stabilization proved above, we can expect that localization error will not disrupt the gradient as long as the coordinates are low-pass filtered to change more slowly than the convergence time.

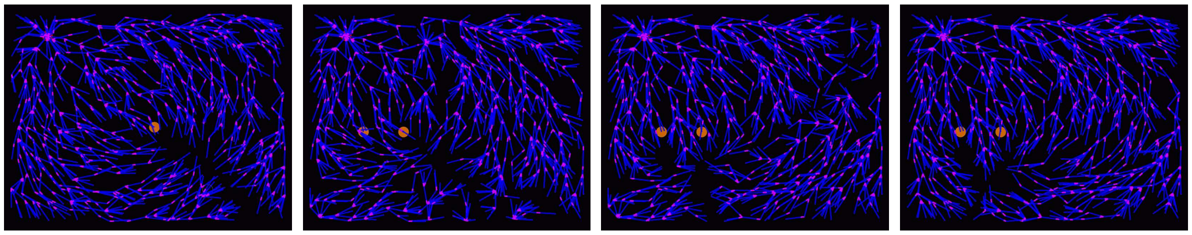
Reception range was software-limited to 9 inches (producing a 7-hop network) in order to allow reliable monitoring of a multi-hop network through a single base-station. The length of a round was set to 1 second, and we set the exponential back-off rate to $k = 1.6$ and the maximum delay to $M = 1.6^{11}$—one transmission every 176 seconds.

We verify that the algorithm behaves as expected by comparing the estimates of distance to the straight-line distance to the source. The esti-



(a) T=0 (b) T=11 (c) T=31 (d) T=74

Figure 7: **CRF-Gradient** reconfigures in response to a change of source location (orange), running in simulation on a network of 1000 devices, 19 hops across. The network is viewed at an angle, with the value shown as the height of the red dot above the device (blue). Reconfiguration spreads quickly through areas where the new value is lower than the old (b), then slows in areas where the new value is significantly higher (c), completing 74 rounds after the source moves.



(a) T=0 (b) T=21 (c) T=41 (d) T=90

Figure 8: A threat avoidance program using **CRF-Max-Probability** reconfigures in response to a change in threat location (orange), running in simulation on a network of 1000 devices, 19 hops across. The vectors from each device display the estimated minimum-threat path from that device towards the destination (red). Reconfiguration is slower because the critical path bends around the outer edge of the network, completing 90 rounds after the threat moves.

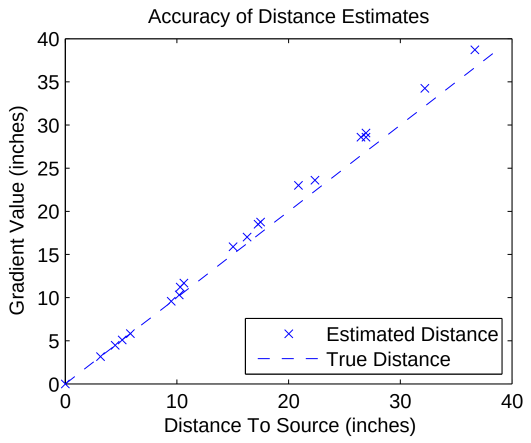


Figure 10: **CRF-Gradient** calculates good range estimates on our test network. The high quality of the estimates is unsurprising, given good connectivity and perfect range data, but serves to confirm that **CRF-Gradient** is behaving as expected.

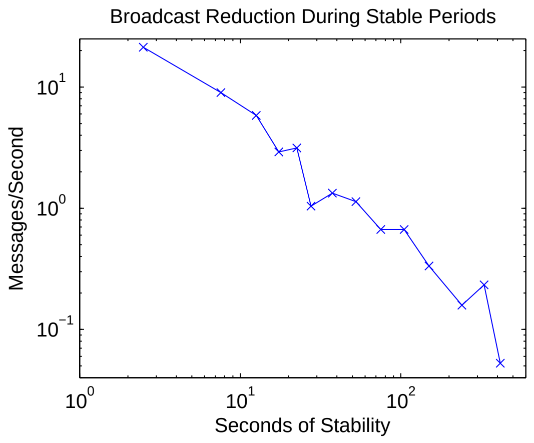


Figure 11: When the values of **CRF-Gradient** are not changing, the discrete implementation automatically throttles the broadcast rate back exponentially. In our experiments, we used a back-off rate of $k = 1.6$, to a maximum delay of $M = 1.6^{11}$—once every 176 seconds. When a source is added to a 20-device network 7 hops wide with no sources, there is a brief flurry of activity followed by a steady back-off to a final rate of less than 1% of the peak.

mates calculated by **CRF-Gradient** are unsurprisingly accurate, given the mesh-like layout and synthetic coordinates (Figure 10). The error in the estimates is entirely due to the difference between the straight-line path and the straightest path through the network.

Of more interest is the ability of **CRF-Gradient** to stabilize to an energy efficient state. We tested broadcast throttling by allowing the network to become unconstrained, then adding a source at one edge of the network. There is an initial flurry of messages as constraint from the source propagates through the network, followed by a steady back-off to a final transmission rate of less than 1% of the peak approximately five minutes later (Figure 11).

7 Contributions

We have introduced Constraint and Restoring Force framework and demonstrated that it can be effective for maintaining scalar fields on sensor networks. When used to calculate gradients, the framework produces a self-stabilizing algorithm that adapts quickly to changes, yet transmits rarely when quiescent. CRF can also be applied to other problems, such as probability calculations for threat avoidance.

This approach appears applicable to a wide variety of problems, potentially creating more robust versions of existing algorithms and serving as a building block for many sensor network applications. Other potential future extensions include damping of small transients and a cleaner continuous time formulation of CRF.

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. Programming an amorphous computational medium. In *Unconventional Programming Paradigms International Workshop*, September 2004.

[4] J. Beal. Amorphous medium language. In *Large-Scale Multi-Agent Systems Workshop (LSMAS)*. Held in Conjunction with AAMAS-05, 2005.

[5] J. Beal and J. Bachrach. Infrastructure for engineered emergence in sensor/actuator networks. *IEEE Intelligent Systems*, pages 10–19, March/April 2006.

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

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

[8] A. Eames. Enabling path planning and threat avoidance with wireless sensor networks. Master's thesis, MIT, June 2005.

[9] 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.

[10] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of Programming Language Design and Implementation (PLDI) 2003*, June 2003.

[11] R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using *airos*. In *DCOSS*, pages 126–140, 2005.

[12] A. Howard, M. J. Mataric, and G. S. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Auton. Robots*, 13(2):113–126, 2002.

[13] 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.

[14] 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.

[15] R. Newton and M. Welsh. Region streams: Functional macroprogramming for sensor networks. In *First International Workshop on Data Management for Sensor Networks (DMSN)*, Aug. 2004.

[16] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil. Distributed, physics-based control of swarms of vehicles. *Auton. Robots*, 17(2-3):137–162, 2004.

[17] D. Vail and M. M. Veloso. Multi-robot dynamic role assignment and coordination through shared potential fields. In A. Schultz, L. Parkera, , and F. Schneider, editors, *Multi-Robot Systems*, pages 87–98. Kluwer, 2003.

[18] 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.