Optimally-Self-Healing Distributed Gradient Structures through Bounded Information Speed

Optimally-Self-Healing Distributed Gradient Structures through Bounded Information Speed*

Giorgio Audrito[1,2], Ferruccio Damiani[1,2], and Mirko Viroli[3]

[1] Dipartimento di Informatica, University of Torino, Torino, Italy
[2] Centro di Competenza per il Calcolo Scientifico, University of Torino, Torino, Italy
{giorgio.audrito,ferruccio,damiani}@unito.it
[3] DISI, University of Bologna, Cesena, Italy
mirko.viroli@unibo.it

Abstract. With the constant increase in the number of interconnected devices in today networks, more and more computations can be described by *spatial computing* abstractions. In this context, distances can be estimated in a fully-distributed way by the so-called *gradient* self-organisation pattern: it is a basic building block also for large-scale system coordination, frequently used to broadcast information, forecast pointwise events, as carrier for distributed sensing, and as combinator for higher-level spatial structures. However, computing gradients is very problematic in a mutable environment: existing algorithms fail in reaching adequate trade offs between accuracy and reaction speed to environment changes.
In this paper we introduce a new gradient algorithm, *BIS (Bounded Information Speed) gradient*, which uses time information to achieve a smooth and predictable reaction speed, which is proved optimal for algorithms following a single-path-communication strategy. Following a proposed methodology for empirical evaluation of performance of spatial computing algorithms, we evaluate BIS gradient and compare it with other approaches. We show that BIS achieves the best accuracy while keeping smoothness under control.

Keywords: Aggregate Programming · Gradient · Information Speed · Reliability · Spatial Computing

1 Introduction

The increasing availability of computational devices of every sort, spread throughout our living and working environments, is creating new challenges in the engineering of complex software systems, especially in contexts like the Internet-of-Things, Cyber-Physical Systems, Pervasive Computing, and so on. *Spatial computing* abstractions have been proposed as a means to take full opportunity of

* This work has been partially supported by: EU Horizon 2020 project HyVar (www.hyvar-project.eu), GA No. 644298; ICT COST Action IC1402 ARVI (www.cost-arvi.eu); Ateneo/CSP D16D15000360005 project RunVar (runvar-project.di.unito.it).

such large-scale computational infrastructures, for their ability to provide pervasive and intelligent sensing, coordination, and actuation over the physical world [5]: they provide models and mechanisms raising the abstraction layer, making it possible to more easily capture the goal of a large-scale situated system.

In this context, "collective" programs can be seen as taking as input situated data changing over time, typically perceived by (virtual or physical) sensors [9], and produce analogous data as outputs to feed (virtual or physical) actuators, having an effect on other computational components, on the physical world, or on humans in it. Such an input/output transformation is captured by a computational process iteratively executing over space and time, involving complex coordination patterns, and in need of satisfying multiple non-functional requirements: scalability, resilience to unpredictable changes, and heterogeneity and dynamism of the communication infrastructure. A key difficulty in engineering collective applications of this kind, hence, is the lack of computational frameworks and libraries of reusable algorithms with guaranteed resilience and performance to match this level of complexity in application services. Even the most basic "building block" algorithms one wants to rely upon, typically give rise to inadequate behaviour when faced with such demanding requirements.

A prototypical example of this phenomenon is given by the *shortest path* (SP) problem in a weighted network, which is fully solved in a traditional computation setting by (among many) Dijkstra's algorithm. In spatial computing, the SP problem translates into the so-called *gradient* computation [4], which amounts to computing shortest paths from all nodes to a given set of source nodes, through a fully distributed process to be iteratively executed to promptly react to any change in the environment. Gradients are known to be a basic building block for self-organising coordination [3, 15, 20, 30], being frequently used for a variety of purposes: to broadcast information, forecast events, dynamically partition networks, ground distributed sensing [6], anticipate future events [21], and to combine into higher-level spatial structures [15]. However, the known algorithms for gradient computation are not fully satisfactory, as they involve relevant trade-offs between scalability, resiliency and precision.

In this paper we introduce a new gradient algorithm, the *BIS (Bounded Information Speed) gradient*, which highly relies on time information to achieve smooth reaction to changes with predictable speed. Given a rising speed v (i.e., increase in gradient estimate over time) as a parameter, it enforces an information propagation speed (i.e., space travelled by information over time) equal to v, so as to scale from the classic gradient (where essentially $v = 0$) to a reaction speed that we prove to be optimal (among the single-path communication algorithms) with v equal to the average information speed. If v is greater than such an average, however, a metric distortion is induced that causes the algorithm to systematically overestimate gradient values: it is thus crucial to tune correctly the parameter in order to achieve the best accuracy.

To address this problem, we compute mathematical estimates of the average single-path communication speed, and use them for validating the performance of BIS gradient with respect to the three most performing algorithms proposed

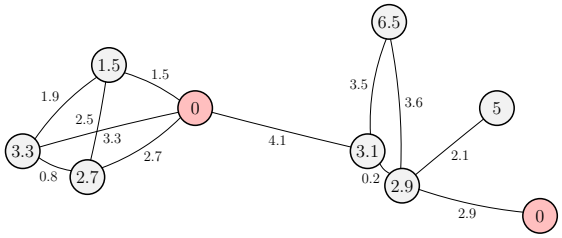

Fig. 1. Gradient computed in a sample network with two source devices.

in the spatial computing context: classic (propagating triangle inequality [30]), CRF [4] and FLEX [3]. We thus show that the BIS gradient achieves the best accuracy while keeping smoothness under control. This comparison is carried out through a general approach we propose, as an empirical evaluation methodology for the performance of all spatial computing algorithms (and gradient algorithms in particular). Finally, we present a realistic case study application of crowd steering towards multiple points of interest (POI), some of which can suddenly become unavailable—faster healing algorithms are here needed to reduce the "average travelling time" for people towards an available POI.

The remainder of this paper is organised as follows. Section 2 provides the background for this paper and discusses related works, introducing the relevant gradient algorithms. Section 3 describes the proposed BIS gradient algorithm together with the mathematical estimates of average single-path information speed. Section 4 proposes the methodology for empirical evaluation of spatial computing algorithms, compares the various gradient algorithms and tests them in the selected case study. Section 5 concludes and outlines possible directions of future research.

2 Background and Related Work

2.1 Gradient-based approaches

In this paper we are concerned with coordination strategies for situated networks, where the objective can be represented in terms of a global, system-level "pattern" to be achieved by local interactions between neighbouring devices, showing inherent resilience with respect to unpredicted changes—in network topology, scale, inputs coming from sensors, and so on. This viewpoint is endorsed by a number of works in a recent thread of research, in the context of coordination models and languages [20, 28, 32], multiagent systems [10, 14, 29] and spatial computing [2, 5, 6, 15, 17]. In spite of various differences and peculiarities, they all promote the idea of creating complex distributed algorithms as *spatial computations*, where few basic communication and coordination mechanisms are provided to the programmer, who uses them by progressively stacking building blocks into layers of increasing complexity.

In this context, gradient data structures, or *gradients* for short, are pervasively used as key building blocks [6, 15]. They produce a map – also called a *computational field* [12, 13] – assigning to each device δ in a network N its estimated

distance from the closest *source* device (an input for the problem), computed by the shortest-path through weighted links in the network (see Fig. 1).

Applications of gradients are countless. Other than to trivially estimate long-range distances (possibly according to metrics computed during execution of the algorithm), gradient computations enact an outward progressive propagation of information along optimal paths. Thus, they are used as forward "carrier" for broadcasting information, forecasting events, and dynamically partitioning networks [6]. Also, used backwards, one can make information flow back to the source, to move or steer mobile agents or data towards the source, or to summarise or average distributed information, i.e., to generally support distributed sensing [6]. Other applications include: considering future events so as to provide proactive "adaptation" [21]; managing semantic knowledge in situated environments [16], create high-level spatial structures [15], elect leaders on a spatial basis [7], and so on.

Due to their usefulness, several works also study how to establish gradients in contexts where local estimation of distances is not available [18, 19, 22], and others take them as basic example to study self-stabilisation techniques [11, 20].

2.2 Gradient-based implementations

According to the framework presented in [27], it is suggested to associate to fundamental building blocks (including the gradient) a library of alternative implementations, among which one has to pick the right implementation for each specific use in the application at hand. It is therefore of interest to analyse different trade-offs in the implementation of gradient algorithms, with the goal of identifying approaches guaranteeing reactivity and smoothness in the way gradients (and the many applications on top) can respond to dynamic environments. In its most basic form, the gradient can be calculated through iterative application of a triangle inequality constraint in each device δ, starting with ∞ everywhere:

$$G(\delta) = \begin{cases} 0 & \text{if } \mathit{source}(\delta) \\ \min\{G(\delta') + w(\delta', \delta) : \ \delta' \in N \text{ linked with } \delta\} & \text{otherwise} \end{cases}$$

We call this procedure *classic* gradient. Repeated fair application[4] of this calculation in a fixed network will converge to the correct value at every point [11].[5] However, the performance of this algorithm in a mutable environment is impaired by several limitations.

- *Speed Bias*: if devices are continuously moving, the values produced by the algorithm systematically underestimate[6] the correct value of the gradient; with an error which increases with the movement speed.

[4] A sequence of updates is *fair* if every device updates his value infinitely often.

[5] In finite time if every device can reach a source; at the limit otherwise (in this case, distance estimates raise indefinitely towards the correct value ∞).

[6] Sporadic overestimation is also possible, however the "minimising" nature of the algorithm propagates lower estimates and disperses higher ones.

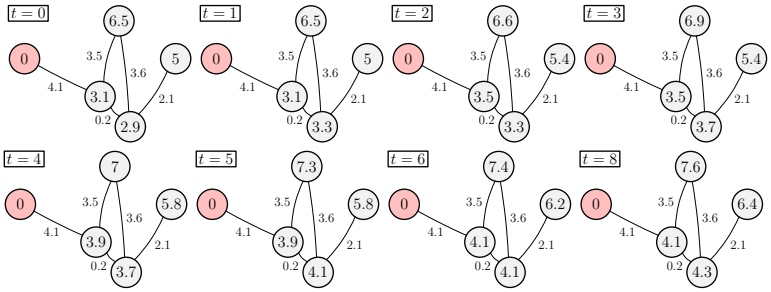

Fig. 2. Evolution after loss of the right source. In each round all devices compute in order, from the one holding the highest value to the one with the lowest. Each device rises by 0.4 every two rounds, because of the short link at the middle of the graph.

- *Rising Value*: in response to quick changes in the network (e.g., a change in the set of source devices), the algorithm can rapidly correct values that need to drop, while it is very slow in correcting values that need to rise. In other words, the algorithm can badly underestimate values for long periods of time after such changes. Precisely, the rising speed of this algorithm is bounded by the distance between the pair of closest devices: Fig. 2 shows an example of this phenomenon on a part of the network in Fig. 1. This problem is also known as *count-to-infinity* in the context of routing algorithms [25].
- *Smoothness*: in the presence of error in distance estimates, it might be preferable not to strictly follow the triangle inequality, so as to reduce the resultant flickering in the output values. Moreover, if the distance estimates are used for an higher-order coordination mechanism (e.g., for moving values towards the sources by "descending" the shortest-paths tree obtained from the gradient), then each variation in the estimates might change the resulting connection tree, effectively disrupting the outcome of the coordination for some time.

In order to overcome these limitations, several refined algorithms have been proposed. To the best of our knowledge, those that better address those problems are J. Beal's CRF gradient (Constraint and Restoring Force) [4] and FLEX gradient (Flexible) [3].

CRF Gradient. The CRF gradient [4] is designed to address the rising value problem by ignoring some *Constraints* (i.e., neighbours[7]), while assuming a *Restoring Force* inducing a uniform rise in absence of constraints. The algorithm takes as parameter a fixed speed v_0, and associates a "rising speed" $v(\delta)$ to each device so that: if the value of the device is currently constrained (either by being a source or by the value of some neighbour) then $v(\delta) = 0$; otherwise if the value is *not* constrained (i.e., all neighbours have been discarded) then $v(\delta) = v_0$. Before applying the minimisation as in the classic gradient, the CRF gradient considers

[7] Recall that in the classic gradient, the value $G(\delta)$ is obtained by combining the "triangle-inequality" constraints $G(\delta) \leq G(\delta') + w(\delta', \delta)$ for each neighbour δ'.

a neighbour δ' as "able to exert constraint" if and only if:

$$G(\delta') + w(\delta', \delta) \leq G(\delta) - \lambda(\delta', \delta) \cdot v(\delta)$$

where $\lambda(\delta', \delta)$ measures time lag, i.e., how old is the information in δ about δ'. The above condition checks whether the constraint given by δ' is able to bound the currently (i.e., not yet updated) value of the gradient *as shifted back to the time when the constraint was calculated*. If the current device is not yet rising, the condition amounts to the constraint being able to reduce the current value; otherwise it becomes more restrictive.

If some neighbour able to exert constraint exists, the value is calculated similarly to the classic gradient. Otherwise, a fixed rising speed is enforced (thus rising by $v_0 \Delta t$ where Δt is the time interval between the last two rounds).

$$G(\delta) = \begin{cases} 0 & \text{if } \mathit{source}(\delta) \\ \min\{G(\delta') + w(\delta', \delta) : \ \delta' \text{ exerts constraint}\} & \text{if some } \delta' \text{ exists} \\ G(\delta) + v_0 \Delta t & \text{otherwise} \end{cases}$$

Through this algorithm the rising speed is then equal to v_0, provided that v_0 is small enough, thus addressing the rising value problem.

FLEX Gradient. The FLEX gradient [3] is designed to improve smoothness through application of a "filtering function" to the outcome of the minimisation, which reduces changes while granting an overall error of at most a given parameter ϵ. Precisely, it first calculates the "maximum local slope":

$$s(\delta) = \max \left\{ \frac{G(\delta) - G(\delta')}{w(\delta', \delta)} : \ \delta' \in N \text{ linked with } \delta \right\}$$

This slope is then used to calculate the gradient estimation as:

$$G(\delta) = \begin{cases} 0 & \text{if } \mathit{source}(\delta) \\ G(\delta') + (1+\epsilon) w(\delta', \delta) & \text{if } s(\delta) > 1 + \epsilon \\ G(\delta') + (1-\epsilon) w(\delta', \delta) & \text{if } s(\delta) < 1 - \epsilon \\ G(\delta) & \text{otherwise} \end{cases}$$

where δ' is the device achieving maximum slope (according to the values available to the current device). The above formula, in other words, selects the closest value to $G(\delta)$ in the interval from $G(\delta') + (1-\epsilon) w(\delta', \delta)$ to $G(\delta') + (1+\epsilon) w(\delta', \delta)$, thus attempting to reduce local changes as much as possible while introducing a metric distortion below ϵ. Two further optimisations are also introduced in FLEX gradient: first, the classical gradient formula is used instead of the above one whenever the current value is over a factor 2 from the old value, or anyway every once in a while (details can be found in [3])—this prevents a systematic error of ϵ to persist indefinitely in a static environment after a network change; second, a distorted metric $w'(\delta', \delta) = \max(w(\delta', \delta), k)$ is used, for a certain constant k—this adds some further error in the output of the algorithm, but it also ensures that the rising speed is at least k (since k becomes the shortest possible "distorted" distance between devices).

3 BIS Gradient

3.1 Information Speed

In most spatial computing abstractions, a network of devices typically perform an interaction through short-range message passing between neighbour devices. The speed achieved by information in this process constitutes an upper bound for responsiveness to environment and input changes, in a similar way to the speed of light, which is an upper bound for causal relationship between events. Depending on the pattern followed by information exchanges, we can distinguish between two main achievable speeds: *single-path* and *multi-path*.

Definition 1 (Information Speed) *The* single-path information speed *is the space travelled over time by messages through a spanning tree in the network. The* multi-path information speed *is the same quantity assuming messages are exchanged through all possible links in the network.*

Clearly, the upper bound for causal relationship in a network is given by the multi-path information speed. This communication pattern requires multiple informations to be aggregated in each node, in order to avoid a program state explosion; and is thus typical of "aggregation" algorithms (such as broadcasting and collecting). Conversely, communication in existing gradient algorithms (and in particular in the BIS gradient we shall introduce in the next subsection) is usually structured on an implicit (shortest-paths) spanning tree: messages from all neighbours are received, but only one of them is selected and passed over for subsequent computations. For this reason, in the remainder of this section we shall focus on single-path information speed and estimate its average v_{avg} in a random network. This estimate would be crucial to determine the value to be passed to BIS gradient for its parameter v.

Consider a network of computing devices, each of them running an algorithm with a certain time period P on data available from neighbour devices within a certain radius R. Let D be a random variable for the distance and T for the time interval between the event of a device sending a message and the event of another device using that message for computation. Then the average speed S achieved by information can be expressed as:[8]

$$E(S) = E\left(\frac{D}{T}\right) = \frac{E(D)}{E(T)}\left(1 + \frac{V(T)}{E(T)^2}\right)$$

truncating the bivariate Taylor expansion of the ratio function to the second order (see [26] for a complete proof of this fact).

The average distance crossed by a message can be calculated as the average radius of communication R times the average distance of a uniformly chosen random point in an n-dimensional unit ball, giving a total $E(D) = \frac{n}{n+1}R$. If devices are moving at a certain average speed v, this estimate should be adapted

[8] Following standard statistic notation, we use $E(X)$ for the mean and $V(X)$ for the variance of a random variable X.

to take into account that messages with a certain lag T could come from a further distance up to vT. An exact calculation of the expected distance in this setting is complex and depends on many factors. However, if we assume the movement to be sufficiently uniform and restrict ourselves to algorithms with a preference for shortest-paths (as for gradients), we can just add up $\frac{v}{2}T$ to the numerator obtaining a roughly acceptable estimate $E(S) = E\left(\frac{D}{T}\right) + \frac{v}{2}$.

The average time interval between events depends heavily on the underlying specific implementation of network communication. In many spatial computing models, like field calculus [31] and Proto [2], a reasonable model of time delay would be $T = P \cdot (I + IF)$, where: P represents the period of a random device, I represents the imprecision of a single device, F represents a random phase between devices, as a uniform distribution of values in $[0, 1]$. In this model, $E(T) = \frac{3}{2}E(P)E(I) = \frac{3}{2}Q$ (where Q is the average computation period) and:

$$1 + \frac{V(T)}{E(T)^2} = \frac{E(T^2)}{E(T)^2} = \frac{E(P^2)}{E(P)^2} \cdot \frac{E\left((I+IF)^2\right)}{E(I+IF)^2} = \frac{E(P^2)}{E(P)^2} \cdot \left(1 + \frac{V(I+IF)}{E(I+IF)^2}\right)$$
$$= \frac{E(P^2)}{E(P)^2} \cdot \left(1 + \frac{V(I) + \frac{V(I)}{3} + \frac{E(I)^2}{12}}{\frac{9}{4}E(I)^2}\right) = \left(1 + \frac{V(P)}{E(P)^2}\right) \cdot \left(\frac{28}{27} + \frac{16}{27}\frac{V(I)}{E(I)^2}\right).$$

Notice that $\frac{V(X)}{E(X)^2}$ is the square of the relative standard error $\hat{\sigma}^2(X)$. Thus the average single-path information speed v_{avg} can be estimated as:

$$v_{\text{avg}} = E(S) = \frac{2}{3}\frac{n}{n+1}\frac{R}{Q}\left(1 + \hat{\sigma}^2(P)\right)\left(\frac{28}{27} + \frac{16}{27}\hat{\sigma}^2(I)\right) + \frac{v}{2} \tag{1}$$

where v is the movement speed of devices. This equation tells us that: the speed is mainly proportional to the ratio of communication radius over computation period; the speed increases with the dimensionality of the space, i.e., is lower for devices aligned in a row and higher for devices in 3-dimensional space;[9] the speed increases with the relative error of computation periods, both among different devices and inside a single device. Equation 1 will be used later to estimate the v parameter of the BIS gradient algorithm. We remark that the average above is computed for a single hop of communication. Over multiple hops, the relative standard error decreases while the average does not change significantly. In case the network parameters (average radius of communication, computation period, etc.) cannot be assumed to be constant, a simple algorithm can still estimate v_{avg} continuously according to the formula above (by averaging the relevant quantities through low-pass filters).

3.2 Computing Gradient through Information Speed

As exemplified in Fig. 2, in presence of a rising value problem, distance increase per round is bounded by the shortest link in the network ℓ. We accordingly

[9] Gradient algorithms have preference for shortest-path links, so that information tends to propagate linearly regardless of the dimensionality of the space. This fact does not contradict the above estimate, which assumes that transmission links are chosen randomly (assumption viable also for gradient algorithms in sparse networks).

obtain an average information speed proportional to $\frac{2\ell}{3Q}$ instead of $\frac{2nR}{3(n+1)Q}$, which can be arbitrarily slower as ℓ approaches zero. This fact suggests us to prevent the rising value problem by *lower bounding* the information speed to make this "slow" rise impossible.

The *Bounded Information Speed* (BIS) gradient improves over the classical gradient by enforcing a minimum information speed v requested by the user. As long as v does not surpass the average single-path communication speed, the algorithm is able to compute correct estimates of the gradient with increased responsiveness. Greater values of v induce instead a metric distortion, causing the algorithm to systematically overestimate values. In the remainder of this paper, we shall thus express v as a fraction of v_{avg} (the average single-path communication speed, which we estimate through Equation 1).

For each device in the network, we compute both the usual gradient estimate $G(\delta)$ and a lag estimate $L(\delta)$, representing the time elapsed since the message started from a source. Lags are estimated through local time differences, so that no overall clock synchronisation is required. When considering a candidate neighbour δ' of a device δ, the time lag relative to this neighbour is:

$$L(\delta, \delta') = L(\delta') + \lambda(\delta', \delta)$$

where $\lambda(\delta', \delta)$ is the lag of the message from δ' to δ. We then take into account this value when calculating the gradient estimate relative to this neighbour:

$$G(\delta, \delta') = \max\{G(\delta') + w(\delta', \delta),\ vL(\delta, \delta') - r\}$$

where w is the distance between devices and r is the communication radius. This formula accounts to assuming that messages propagate at least at speed v, so that the gradient estimate is lower bounded by $vL(\delta, \delta')$ (with the additive constant $-r$ to ensure that some error is taken into account).

The overall estimates of $G(\delta)$ and $L(\delta)$ are then obtained by minimising $G(\delta, \delta')$ over neighbours (we assume that pairs are ordered lexicographically):

$$[G(\delta), L(\delta)] = \begin{cases} [0, 0] & \text{if } \mathit{source}(\delta) \\ \min\{[G(\delta, \delta'), L(\delta, \delta')] :\ \delta' \in N \text{ linked with } \delta\} & \text{otherwise} \end{cases}$$

This algorithm generalises the classic gradient algorithm, as shown in the following.

Theorem 1 (Degenerate BIS) *The BIS gradient with $v = 0$ is equivalent to the classic gradient.*

Proof. If $v = 0$, $G(\delta, \delta') = \max\{G(\delta') + w(\delta', \delta),\ 0 \cdot L(\delta, \delta') - r\}$ is equal to $G(\delta') + w(\delta', \delta)$ so that $L(\delta)$ is implicitly discarded.

In particular, the same result would hold for devices with no lag estimator so that $\lambda(\delta, \delta')$ is always 0. For devices with an internal timer (so that a lag estimator can be defined), tweaking the parameter v close to the average single-path information speed provides a guaranteed reactivity, which is optimal among algorithms with a single-path information flow.

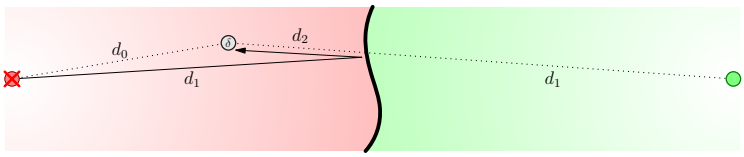

Fig. 3. Information flow upon disconnection of a source device.

Theorem 2 (Performance Bound) *Information speed in BIS gradient, calculated w.r.t. the gradient estimates, is at least v. Furthermore, values constrained by obsolete information increase at least at speed v.*

Proof. Since $G(\delta, \delta') \geq vL(\delta, \delta') - r$ for all δ', also $G(\delta) \geq vL(\delta) - r$ concluding the first part. For the second part, consider an information that started propagating from a certain source at time t_0 and is now obsolete (e.g., the source has been disconnected), and fix a device δ computing in times $t_1, \ldots, t_n$ constrained by such obsolete information. Since $L(\delta) = t_i - t_0$ in each computing round $i \leq n$, $G(\delta) \geq vL(\delta) - r = v(t_i - t_0) - r$ concluding the second part.

Theorem 3 (Optimality) *The BIS gradient with v equal to the average single-path information speed v_{avg} attains optimal reactivity among algorithms with a single-path information flow.*

Proof. As a prototypical example, consider an already stabilised network with a selected source device and its corresponding influence region, i.e., the set of devices whose distances are calculated w.r.t. the selected source (red). Suppose that the selected source device is suddenly disconnected at time $t = 0$. In any algorithm with a single-path information flow, the information about this disconnection flows through the influence region at average speed v. For example, device δ in Fig. 3 is reached by this information at time $\frac{d_0}{v}$, and it cannot change its value from d_0 before that time.

In the best case scenario, after the information about the disconnection reaches the border a new wave of information can bounce back towards the inside of the region, bringing values calculated from other sources (green). Since the shortest path from the disconnected source to the border and then back to device δ has length $d_1 + d_2$ (black arrow) and information flows at speed v, the earliest time when δ can reach the correct value is $\frac{d_1+d_2}{v}$. Notice that this value is $d_1 + d_2$ since the distance from the border to the two sources is the same.

Then δ holds value d_0 at time $\frac{d_0}{v}$ and value $d_1 + d_2$ at time $\frac{d_1+d_2}{v}$, effectively rising at speed $(d_1+d_2-d_0)/(\frac{d_1+d_2}{v} - \frac{d_0}{v}) = v$. Since this is the best-case scenario, a faster rising speed is not possible thus proving optimality of BIS gradient.

3.3 Reducing Volatility and Communication Cost

An improved reactivity to changes naturally translates into an increase in volatility of values, thus reducing the degree of *smoothness*. This holds true also for the BIS gradient: in a mutable environment, even calculating the exact gradient

all the times would perform poorly on smoothness, since it would rapidly adapt all the values as noise and small movements take place.

In order to improve smoothness of rapidly self-healing algorithms, it is then necessary to insert a damping component. Also the FLEX gradient, designed for improved smoothness, can be seen as the embedding of the following damping function into the classical gradient computation:

$$\texttt{damp}(\texttt{old},\texttt{new}) = \begin{cases} \texttt{new} + \epsilon w(\delta',\delta) & \text{if } \texttt{old} > \texttt{new} + \epsilon w(\delta',\delta) \\ \texttt{new} - \epsilon w(\delta',\delta) & \text{if } \texttt{old} < \texttt{new} - \epsilon w(\delta',\delta) \\ \texttt{old} & \text{otherwise} \end{cases}$$

In future works, it is therefore natural to investigate whether the insertion of this damping function (or others) into algorithms other than the classic gradient would achieve the same effect. In the next section, we shall show that this is true to some extent, allowing the BIS gradient for an improved smoothness.

4 Analysis and Verification

4.1 Performance Indicators

In order to empirically evaluate the performance of an approximated localised algorithm,[10] several aspects need to be taken into account. We divide them into *environment* characteristics, *input* properties, and *output* requirements.

Environment. A spatio-temporal computing environment is characterised by its degree of steadiness, which both in time and in space can be further specified through measures of *noise* and *variability*. We classify as *noise* the small high-frequency variations which are not intended to alter the expected output of the algorithm: in space, it corresponds to short-range Brownian movements; in time, it corresponds to random error in the frequency of events (in each device). We classify as *variability* the larger low-frequency variations which are intended to alter the expected output: in space, it corresponds to long-range directional movements; in time, it corresponds to systematic error in the frequency of events (changing between devices or through time).

Input. To assess the performance of an algorithm, we need to split tests into two further possible situations: *constant* input, to isolate and measure the responses to environment variations; *discontinuous* input, where a sudden change happens at a certain point in time, to measure the healing speed of the algorithm.

[10] With *approximated localised algorithm* we denote any spatially-distributed iterative process which aim to approximate a target global *input/output* transformation (taking into account environmental data, as described in Section 1). For example, this is the case for *gradient* algorithms which approximate *shortest-path distances* (output) given a source set and an environmental configuration (input).

Output. Given a test environment and input, we need to measure two different qualities of the output generated by the algorithm: *precision* and *smoothness*. *Precision* is the deviation from the ideal outcome: with a constant input, it measures systematic error (e.g., *speed bias* for gradient algorithms); with a discontinuous input, it measures healing speed (e.g., *rising value* for gradient algorithms). *Smoothness* is the volatility of the output values, usually measured as the integral of absolute differences between consecutive values (first derivative of the output), and aims for gradual and unidirectional changes in the output values, absorbing noise. It needs to be measured both on constant and discontinuous input.

Performance assessment of approximated localised algorithms thus requires extensive testing over several different environments, combining diverse degrees of noise and variability (both in space and in time). Among the different possibilities, we recommend to include: zero-noise zero-variability (in both space and time), in which the basic *self-stabilisation* property[11] is measured [11]; high-noise high-variability (in both space and time), in which a bottom line of guaranteed performance is measured in an extreme case; further intermediate cases, which can help differentiate how performance is affected by the different types of mutability (in space or time, as noise or variability), depending on the specific application. In each of those scenarios, performance is measured through *precision* and *smoothness*; on an input which is first constant for a long enough period of time to reach stable results, and then change discontinuously and keeps the new value constant until stable results are reached again.

4.2 Comparison between Gradient Algorithms

In order to compare the performance of the different gradient algorithms presented in this paper, we chose an environment able to trigger the issues presented in Section 2:

- *speed bias*, by considering environments with increasing variability in space;
- *rising value*, through arranging devices densely into a long corridor with a source at one end, so that the ratio between the longest and shortest distance between devices is high;
- *smoothness*, by measuring it in each test scenario.

Following the guidelines introduced in Section 4.1, we thus tested the following scenarios.

- *Environment:* we put 1000 devices with communication radius $10m$ and average fire rate $1s$ randomly into a $500m \times 20m$ corridor, producing a network 50 hops wide. We tested this environment with increasing *variability* in space (long range movements) from 0 (none), to 1 (moderated) and 3 (high). Several conducted tests revealed that *noise* (both in space and in time) and

[11] An algorithm is *self-stabilising* if given a constant environment, it eventually reaches a correct output for any possible initial state.

variability in time did not affect significantly the behaviour of any of the considered algorithms, witnessing their intrinsic robustness. We thus only report graphs with high *noise* and time *variability* (Brownian motion and 50% relative standard error in fire rate between different devices plus another 50% in each device). We modelled randomly distributed events according to a Weibull distribution [33].

- *Input:* we provided the algorithms with a single source, steadily located on the left end of the corridor until time 300, and then abruptly moved to the opposite right end. In this way, reaction to *discontinuous* input is measured (in the middle of the graphs) as well as behaviour under constant input (at the sides of the graphs).
- *Output:* for each scenario we measured *precision* as absolute error w.r.t. Euclidean distance and *smoothness* as absolute difference between values in consecutive rounds (both averaged).

Figure 4 summarises the evaluation results, which were obtained (similarly also to the experiments in next subsection) with Protelis [24] (an incarnation of the Field Calculus [13]) as programming language to code the model, Alchemist as simulator [23] and the Supercomputer OCCAM [1] to run the experiments. We tested classic, CRF, FLEX, BIS with $v = 0.5v_{\text{avg}}$, BIS with $v = 0.5v_{\text{avg}}$ and FLEX damping, BIS with $v = 0.9v_{\text{avg}}$, BIS with $v = 0.9v_{\text{avg}}$ and FLEX damping. In all cases, the tolerance of the FLEX damping was set to 10%. We run 10 instances of each scenario with different random seeds and averaged the results.

The rising value problem corresponds to the spikes in the middle of the graphs, which are considerably shorter (faster healing) for BIS gradient, even when $v = 0.5v_{\text{avg}}$. Speed bias is visible from the increase in error baseline under increasing space variability, and is more contained for BIS gradient (in particular when v is high). The only setting where BIS does not achieve the best precision is under constant input and zero space variability, where the error value is still small and in fact determined by the small variations reported in smoothness.

As expected, the increase in precision corresponds to a decreased smoothness, so that BIS gradient has the highest value volatility (increasing with v). Embedding the FLEX damping into BIS proves to be effective in reducing fluctuations for all values of v, so that BIS with $v = 0.5v_{\text{avg}}$ and FLEX damping score better than CRF gradient and comparably similar to FLEX and classic gradients, while still achieving a much higher precision.

Overall, these results prove that BIS gradient achieves a much higher healing speed and accuracy (especially when v is high), while still keeping smoothness under control (especially when FLEX dumping is also used). This properties are readily appreciable in practical applications where inputs cannot be assumed to be constant: as we shall show in the next subsection, in these settings BIS gradient remains effective whereas other gradient algorithms fail to produce sensible results, disrupting the higher-order coordination mechanisms relying on them.

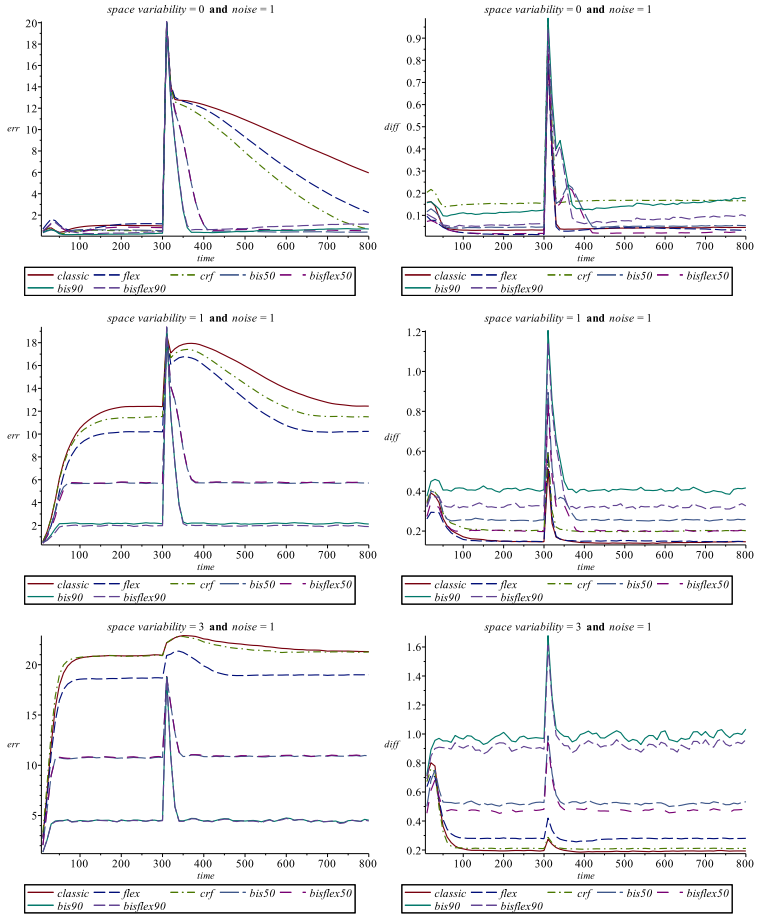

Fig. 4. Precision (left) and smoothness (right) of gradient algorithms under increasing space variability (from top to bottom) and high noise and time variability.

4.3 Case Study: Crowd Steering to Busy Resources

We now compare different gradient algorithms when used to implement a more complex service in a realistic scenario: classic gradient, CRF gradient, FLEX gradient and BIS gradient with $v = 0.9v_{\text{avg}}$. We considered a *crowd steering* application run on people's smartphones and local device-to-device interaction. This service gives directions towards points-of-interest which can be invalidated: e.g., an application steering pedestrians in a large exhibition center with several food stands (restaurants), where it may happen that a restaurant suddenly becomes full. We considered a $200m \times 300m$ fair ambient, containing a 7×2 grid of $20m \times 40m$ obstacles (pavilions). On the four corners of the fair, we located restaurants of random capacity (up to 1000 people). The simulation encompassed 3000 rounds where 3000 individuals wandered randomly with a connection range of $10m$. We assumed that people get hungry (thus starting to

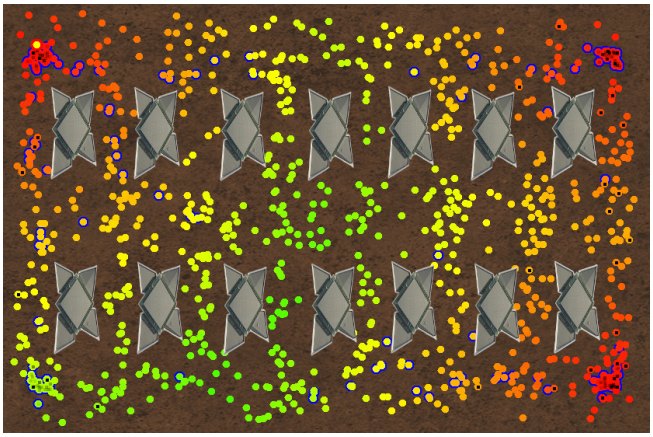

Fig. 5. A screen shot of the application scenario. The restaurant at the bottom left corner is full and the application is steering hungry (blue) people towards the other ones (as reflected by the colour hue, which is determined by the corresponding gradient).

look for a restaurant) at a random round between 50 and 1000; and that after reaching a free restaurant, they stay there for up to 1000 rounds (eating) before restarting wandering again. A 20% relative standard error in fire rate was taken into account, both between different devices and among subsequent rounds on the same device. We considered a relatively low average fire rate ($5s$), in order to emphasise the differences in performance of the different gradient algorithms. We run 10 instances of each scenario with different random seeds and averaged the results. A screen shot of this scenario is presented in Fig. 5.[12]

In this scenario, reactivity of the underlying gradient implementation to input changes (restaurants getting full) was proven to be crucial. Slow reactivity resulted in people reaching busy restaurants, and then waiting there until some place was let free (effectively defeating the purpose of the crowd steering application). Fast reactivity allowed people to direct themselves directly towards restaurants with free places, reducing significantly the waiting time. This behaviour is clearly pictured in Fig. 6, both by the ratio of people reaching a restaurant over time and by the maximum waiting time. The average waiting time, instead, seems to suggest a smaller difference among the performance of the different algorithms. This is due to the fact that only people with a determined waiting time are considered, i.e., which have already reached a restaurant. Thus the average waiting time is expected to keep rising until all the people have already reached a restaurant (which is far from happening for CRF and classic gradient).

Overall, these results prove that using an inefficient gradient algorithm (as e.g., classic gradient) can result in a final real-world application being unusable

[12] For the sake of reproducibility, all the experiments made in this paper are available at `https://bitbucket.org/gaudrito/experiment-fast-gradient`.

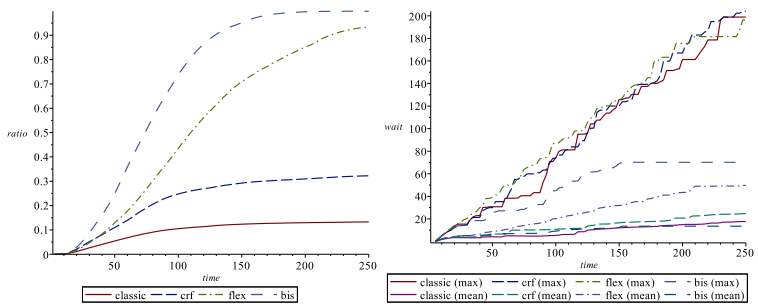

Fig. 6. Percentage of people who found a restaurant over time (left), average and maximum waiting time in minutes (right) for different gradient algorithms.

(as e.g., only 13% of the people were able to eat after 250 minutes in this case). Conversely, using BIS gradient for the underlying gradient routines represents the best choice, outperforming all the other ones by a large amount.

5 Conclusions and Future Works

We have introduced BIS gradient, a new gradient algorithm of optimal self-healing speed among algorithms with a single-path communication scheme. Mathematical estimations to guide the selection of the parameter v are provided. Thorough validation is carried out, both in a realistic case study and in isolation w.r.t. classic, CRF and FLEX gradients, showing the effectiveness of the new algorithm in a variety of contexts. We also use and suggest an empirical evaluation methodology for spatial computing algorithms, seemingly applicable to all *eventually consistent* algorithms [8] and particularly, gradients. In the future, we plan to test the present algorithm on a larger-scale case study with real data.

We believe, however, that there is still some margin for further improvements. For instance, some form of broadcast could be used to surpass the theoretical limit given by single-path communication speed. *Smoothness* could be further improved by fine-tuning other damping functions other than the one given by the FLEX gradient. The *speed bias* could be addressed directly by introducing a metric distortion dependent on the movement speed of devices (in a similar way as it is done in [19]), and several mobility models could be considered to fine-tune both the information speed estimate and the metric distortion. Additionally, it is possible that specific variants of the proposed algorithm can provide additional benefits in specific applications of the gradient pattern; in particular, we are interested in the cases where gradients are used to support distributed sensing of information in highly heterogeneous and dense environments, specifically for crowd engineering applications.

Acknowledgements We thank the anonymous COORDINATION referees for their comments and suggestions on improving the presentation.

References

1. Aldinucci, M., Bagnasco, S., Lusso, S., Pasteris, P., Vallero, S., Rabellino, S.: The Open Computing Cluster for Advanced data Manipulation (occam). In: The 22nd International Conference on Computing in High Energy and Nuclear Physics (CHEP). San Francisco, USA (2016)
2. Bachrach, J., Beal, J., McLurkin, J.: Composable continuous space programs for robotic swarms. Neural Computing and Applications 19(6), 825–847 (2010)
3. Beal, J.: Flexible self-healing gradients. In: Proceedings of the 2009 ACM Symposium on Applied Computing. pp. 1197–1201. SAC '09, ACM (2009)
4. Beal, J., Bachrach, J., Vickery, D., Tobenkin, M.: Fast self-healing gradients. In: Proceedings of ACM SAC 2008. pp. 1969–1975. ACM (2008)
5. Beal, J., Dulman, S., Usbeck, K., Viroli, M., Correll, N.: Organizing the aggregate: Languages for spatial computing. In: Mernik, M. (ed.) Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, chap. 16, pp. 436–501. IGI Global (2013), longer version at: `http://arxiv.org/abs/1202.5509`
6. Beal, J., Pianini, D., Viroli, M.: Aggregate programming for the Internet of Things. IEEE Computer 48(9) (2015)
7. Beal, J., Viroli, M.: Building blocks for aggregate programming of self-organising applications. In: 2nd FoCAS Workshop on Fundamentals of Collective Systems. pp. 8–13. IEEE CS (2014), DOI: 10.1109/SASOW.2014.6
8. Beal, J., Viroli, M., Pianini, D., Damiani, F.: Self-adaptation to device distribution changes. In: Cabri, G., Picard, G., Suri, N. (eds.) 10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2016, Augsburg, Germany, September 12-16, 2016. pp. 60–69 (2016), Best paper of IEEE SASO 2016. DOI: 10.1109/SASO.2016.12
9. Bicocchi, N., Mamei, M., Zambonelli, F.: Self-organizing virtual macro sensors. TAAS 7(1), 2:1–2:28 (2012), DOI: 10.1145/2168260.2168262
10. Castelli, G., Mamei, M., Rosi, A., Zambonelli, F.: Engineering pervasive service ecosystems: The SAPERE approach. TAAS 10(1), 1:1–1:27 (2015)
11. Damiani, F., Viroli, M.: Type-based self-stabilisation for computational fields. Logical Methods in Computer Science 11(4) (2015), DOI: 10.2168/LMCS-11(4:21)2015
12. Damiani, F., Viroli, M., Beal, J.: A type-sound calculus of computational fields. Science of Computer Programming 117, 17 – 44 (2016), DOI: 10.1016/j.scico.2015.11.005
13. Damiani, F., Viroli, M., Pianini, D., Beal, J.: Code mobility meets self-organisation: A higher-order calculus of computational fields. In: Graf, S., Viswanathan, M. (eds.) Formal Techniques for Distributed Objects, Components, and Systems, Lecture Notes in Computer Science, vol. 9039, pp. 113–128. Springer International Publishing (2015), DOI: 10.1007/978-3-319-19195-9_8
14. Elhage, N., Beal, J.: Laplacian-based consensus on spatial computers. In: van der Hoek, W., Kaminka, G.A., Lespérance, Y., Luck, M., Sen, S. (eds.) 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3. pp. 907–914. IFAAMAS (2010)
15. Fernandez-Marquez, J.L., Serugendo, G.D.M., Montagna, S., Viroli, M., Arcos, J.L.: Description and composition of bio-inspired design patterns: a complete overview. Natural Computing 12(1), 43–67 (2013)
16. Fernandez-Marquez, J.L., Tchao, A., Serugendo, G.D.M., Stevenson, G., Ye, J., Dobson, S.: Analysis of new gradient based aggregation algorithms for data-propagation in mobile networks. In: Sixth IEEE International Conference on Self-

Adaptive and Self-Organizing Systems Workshops, SASOW 2012, Lyon, France, September 10-14, 2012. pp. 217–222. IEEE Computer Society (2012)
17. Giavitto, J.L., Michel, O., Cohen, J., Spicher, A.: Computation in space and space in computation. Tech. Rep. 103-2004, Univerite d'Evry, LaMI (2004)
18. Katzenelson, J.: Notes on amorphous computing. In: MIT Artificial Intelligence Laboratory. Citeseer (2000)
19. Liu, Q., Pruteanu, A., Dulman, S.: Gradient-based distance estimation for spatial computers. Comput. J. 56(12), 1469–1499 (2013), DOI: 10.1093/comjnl/bxt124
20. Lluch-Lafuente, A., Loreti, M., Montanari, U.: Asynchronous distributed execution of fixpoint-based computational fields. CoRR abs/1610.00253 (2016), http://arxiv.org/abs/1610.00253
21. Montagna, S., Viroli, M., Fernandez-Marquez, J.L., Di Marzo Serugendo, G., Zambonelli, F.: Injecting self-organisation into pervasive service ecosystems. Mobile Netw Appl 18(3), 398–412 (2013), DOI: 10.1007/s11036-012-0411-1
22. Nagpal, R., Shrobe, H., Bachrach, J.: Organizing a global coordinate system from local information on an ad hoc sensor network. In: Information Processing in Sensor Networks. pp. 333–348. Springer (2003)
23. Pianini, D., Montagna, S., Viroli, M.: Chemical-oriented simulation of computational systems with ALCHEMIST. J. Simulation 7(3), 202–215 (2013)
24. Pianini, D., Viroli, M., Beal, J.: Protelis: Practical aggregate programming. In: ACM Symposium on Applied Computing 2015. pp. 1846–1853 (April 2015)
25. Royer, E.M., Toh, C.: A review of current routing protocols for ad hoc mobile wireless networks. IEEE Personal Commun. 6(2), 46–55 (1999)
26. Stuart, A., Ord, J.K.: Kendall's advanced theory of statistics. Vol. 1. Edward Arnold, London; copublished in the Americas by Halsted Press - John Wiley & Sons, Inc. , New York, sixth edn. (1994)
27. Viroli, M., Beal, J., Damiani, F., Pianini, D.: Efficient engineering of complex self-organising systems by self-stabilising fields. In: Self-Adaptive and Self-Organizing Systems (SASO), IEEE 9th International Conference on. pp. 81–90. IEEE (2015), DOI: 10.1109/SASO.2015.16
28. Viroli, M., Casadei, M.: Biochemical tuple spaces for self-organising coordination. In: Field, J., Vasconcelos, V.T. (eds.) Coordination Models and Languages: 11th International Conference, COORDINATION 2009. Proceedings, pp. 143–162. Springer (2009), DOI: 10.1007/978-3-642-02053-7_8
29. Viroli, M., Casadei, M., Montagna, S., Zambonelli, F.: Spatial coordination of pervasive services through chemical-inspired tuple spaces. ACM Transactions on Autonomous and Adaptive Systems 6(2), 14:1 – 14:24 (June 2011), DOI: 10.1145/1968513.1968517
30. Viroli, M., Damiani, F.: A calculus of self-stabilising computational fields. In: Coordination Languages and Models, LNCS, vol. 8459, pp. 163–178. Springer-Verlag (2014), DOI: 10.1007/978-3-662-43376-8_11
31. Viroli, M., Damiani, F., Beal, J.: A calculus of computational fields. In: Advances in Service-Oriented and Cloud Computing, Communications in Computer and Information Science, vol. 393, pp. 114–128. Springer (2013), DOI: 10.1007/978-3-642-45364-9_11
32. Viroli, M., Pianini, D., Beal, J.: Linda in space-time: an adaptive coordination model for mobile ad-hoc environments. In: Proceedings of Coordination 2012, Lecture Notes in Computer Science, vol. 7274, pp. 212–229. Springer (2012)
33. Weibull, W., et al.: A statistical distribution function of wide applicability. Journal of applied mechanics 18(3), 293–297 (1951)