Logical Methods in Computer Science
Volume 19, Issue 4, 2023, pp. 29:1–29:33
https://lmcs.episciences.org/

Submitted Nov. 01, 2022
Published Dec. 18, 2023

SPACE-FLUID ADAPTIVE SAMPLING BY SELF-ORGANISATION

ROBERTO CASADEI [a], STEFANO MARIANI [b], DANILO PIANINI [a], MIRKO VIROLI [a], AND FRANCO ZAMBONELLI [b]

[a] Alma Mater Studiorum—Università di Bologna, Via dell'Università, 50, Cesena (FC), Italy
e-mail address: roby.casadei@unibo.it, danilo.pianini@unibo.it, mirko.viroli@unibo.it

[b] Università di Modena e Reggio Emilia, Via Giovanni Amendola, 2, Reggio Emilia (RE), Italy
e-mail address: stefano.mariani@unimore.it, franco.zambonelli@unimore.it

Abstract. A recurrent task in coordinated systems is managing (estimating, predicting, or controlling) signals that vary in space, such as distributed sensed data or computation outcomes. Especially in large-scale settings, the problem can be addressed through decentralised and situated computing systems: nodes can locally sense, process, and act upon signals, and coordinate with neighbours to implement collective strategies. Accordingly, in this work we devise distributed coordination strategies for the estimation of a spatial phenomenon through collaborative adaptive sampling. Our design is based on the idea of dynamically partitioning space into regions that compete and grow/shrink to provide accurate aggregate sampling. Such regions hence define a sort of virtualised space that is "fluid", since its structure adapts in response to pressure forces exerted by the underlying phenomenon. We provide an adaptive sampling algorithm in the field-based coordination framework, and prove it is self-stabilising and locally optimal. Finally, we verify by simulation that the proposed algorithm effectively carries out a spatially adaptive sampling while maintaining a tuneable trade-off between accuracy and efficiency.

1. Introduction

A significant problem in computer systems engineering is dealing with phenomena that vary in space: for instance, their estimation, prediction, and control. Concrete related application examples include: the monitoring of waste in urban areas to improve waste gathering strategies [MZ19]; the estimation of pollution in a geographical area, for alerting or mitigation-aimed response purposes [CPP+20]; the sensing of the temperature in a large building, to support the synthesis of control policies for the *Heating, Ventilation, and Air Conditioning (HVAC)* system [MMP+17]. The general solution for addressing this kind of problem consists of deploying a set of sensors and actuators in space, and building a distributed system that processes gathered data and possibly determines a suitable actuation in response [WKT11]. In many settings, the computational activity can (or has to) be performed *in-network* [FRWZ07] in a *decentralised* way: in such systems, nodes locally sense,

Key words and phrases: spatial sampling, cooperative adaptive sampling, regional coordination, sensor networks, field-based computing, self-organisation, event structures, Fluidware.

* This article is an extended version of the conference paper [CMP+22] presented at COORDINATION'22.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.46298/LMCS-19(4:29)2023

process, and act upon the environment, and coordinate with *neighbour* nodes to collectively self-organise their activity. However, in general there exists a trade-off between performance and efficiency, that suggests concentrating the activities on few nodes, or to endow systems with the capability of autonomously adapt the *granularity* of computation [YVP13].

In this work, we focus on sampling signals that vary in space. Specifically, we would like to sample a spatially distributed signal through device coordination and self-organisation such that the samples accurately reflect the original signal and the least amount of resources is used to do so. In particular, we push forward a vision of space-fluid computations, namely computations that are fluid, i.e. change seamlessly, in space and – like fluids – adapt in response to pressure forces exerted by the underlying phenomenon. We reify the vision through an algorithm that handles the shape and lifetime of leader-based "regional processes" (cf. [PCVN21]), growing/shrinking as needed to sample a phenomenon of interest with a (locally) maximum level of accuracy and minimum resource usage. For instance, we would like to sample more densely those regions of space where the spatial phenomenon under observation has high variance, to better reflect its spatial dynamics. On the contrary, in regions where variance is low, we would like to sample the phenomenon more sparsely to, e.g., save energy, communication bandwidth, etc. while preserving the same level of accuracy.

Accordingly, we consider the *field-based coordination* framework of *aggregate computing* [BPV15, VBD+19], which has proven to be effective in modelling and programming self-organising behaviour in situated networks of devices interacting asynchronously. On top of it, we devise a solution that we call *aggregate sampling*, inspired by the approaches of self-stabilisation [VAB+18] and density-independence [BVPD17], that maps an input field representing a signal to be sampled into a regional partition field where each region provides a single sample; then, we characterise the aggregate sampling error based on a distance defined between stable snapshots of regional partition fields, and propose that an *effective* aggregate sampling is one that is locally optimal w.r.t. an error threshold, meaning that the regional partition cannot be improved simply by merging regions. In summary, we provide the following contributions:

- we define a model for distributed collaborative adaptive sampling and characterise the corresponding problem in the field-based coordination framework;
- we implement an algorithmic solution to the problem that leverages self-organisation patterns like gradients [FSM+13, VAB+18] and coordination regions [PCVN21];
- we prove this algorithm to self-stabilise, and to actually provide an effective sampling according to a definition of "locally optimal regional partition";
- we experimentally validate the algorithm to verify interesting trade-offs between sparseness of the sampling and its error.

This manuscript is an extended version of the conference paper [CMP+22], providing *(i)* a more extensive and detailed coverage of related work; *(ii)* more examples, clarifications, and details regarding the formal model; *(iii)* a discussion of the source code of the aggregate computing implementation; and *(iv)* proofs of self-stabilisation and local optimality of the proposed algorithm.

The rest of the paper is organised as follows. Section 2 covers motivation and related work. Section 3 provides a model for distributed sampling and the problem statement. Section 4 describes an algorithmic solution to the problem of sampling a distributed signal using the framework of aggregate computing. Section 5 performs an experimental validation

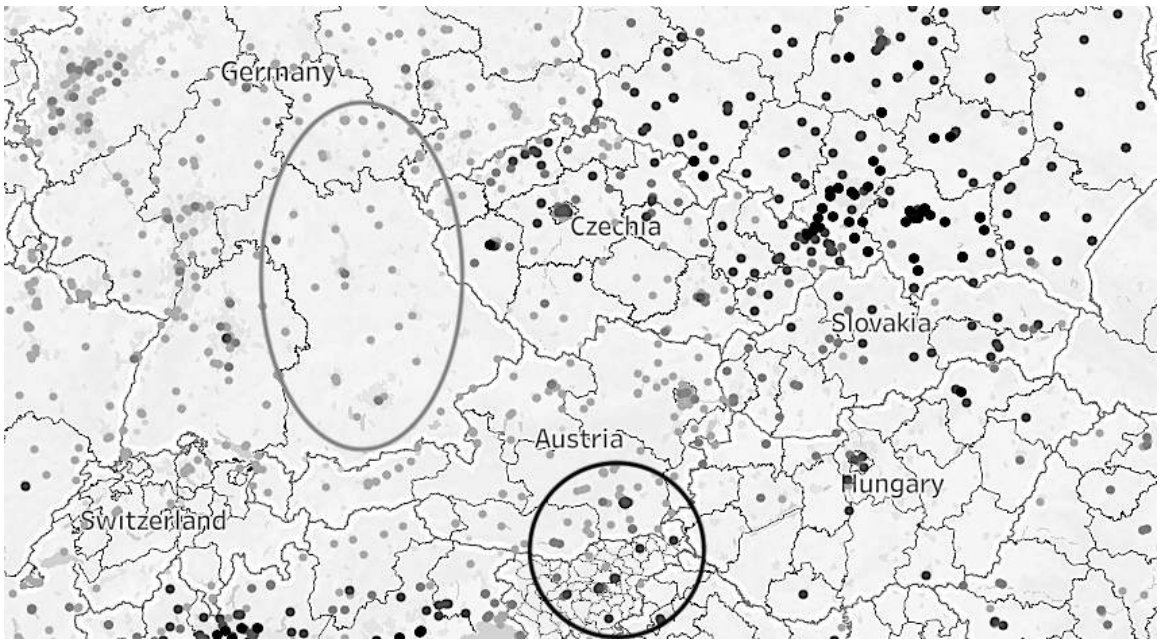


FIGURE 1. Air quality statistics map taken from `https://archive.ph/dMJO2`. There are areas where the underlying phenomenon does not vary significantly in space (light-grey oval), hence sampling could be made sparser with tolerable loss of accuracy. In others (darker circle), variance is high, requiring a more detailed spatial sampling.

of the proposed approach. Finally, Section 6 provides conclusive thoughts and delineates directions for further research.

2. Motivation and Related Work

2.1. **Motivations, Goal, and Applications.** Consider a Wireless Sensor Network (WSN) of any topology, *statically* (i.e. design-time, no mobility) deployed across a geographical area to monitor a spatially-distributed phenomenon, such as, for instance, air quality, as depicted in Figure 1. We want to *dynamically* (at run-time) and *adaptively* (depending on the phenomenon itself) find a sparse set of *samplers*, i.e., devices responsible for providing sensing data regarding some underlying phenomenon. We want the selection of samplers to depend on both the *spatial distribution* of devices and the input phenomenon. Therefore, the idea is that each sampler is responsible for an exclusive *spatial sampling region* that may include several other devices, i.e., a partition of the system/environment. Moreover, we want to determine a partitioning of the connected sampling devices that minimises the number and maximises the size of sampling regions, while preserving as much as possible the underlying information. Hence, in areas with low variance amongst spatially distributed samples' values, we want our regions to be larger, as many samples will report similar values, and hence one sample nicely represents all. Conversely, in areas with high spatial variance between samples, smaller regions are necessary as even proximal samples may have very

different values, hence more samples are required to accurately represent the phenomenon. We also consider *large-scale deployments* with hundreds or thousands of devices.

Accordingly, we aim at designing a *decentralised algorithm* that can:

- dynamically (at run-time, continuously) partition a set of sampling devices into sampling regions;
- consider phenomenon-specific metrics (e.g. variance of the sensors' readings) for deciding how to (re)compute partitions;
- both up-scale and down-scale sampling depending on such metrics;
- do so in a fully distributed way, based solely on local interactions (i.e., within a 1-hop neighbourhood).

A similar algorithm can provide benefits across multiple application domains, as also witnessed by the below described related literature. All forms of environmental monitoring can greatly benefit, for instance, as many phenomena in such a domain are inherently spatially distributed: air quality, water pollution, soil radiation, landslide monitoring, crop growth, and so on [Cox99, YZMC20, ZGMB14, MCP12]. Another field of application is ecological monitoring, such as geolocation of wolf packs or other animals moving habits [PSS+13]. But in general, any application whose goal is to monitoring any measurable phenomenon, with unknown or uncertain spatial dynamics, may find benefits in our proposed approach.

2.2. **Related Work on Adaptive Spatial Sampling.** There are several approaches in the literature that attempt to solve this and similar problems, with heterogeneous techniques, that we collectively refer to under the umbrella term of "adaptive spatial sampling". Amongst these, some [MSM18, Tho90] are restricted to the so-called *sampling design* problem, that is, their concern is to deliver either design-time decision support about where to deploy sensor devices or analytically devise out the best sampling algorithm given domain expertise or infrastructural requirements (most often, residual energy management). Our approach is not directly comparable to these, as we are concerned with run-time adaptation of the sampling process based on domain-specific properties (e.g. sampling variance, that depends on the phenomenon under observation). Others [RHK+05, GA14, MHD21, HP11, GC09] more closely pursue our goal, but assume mobile sensing devices (e.g. robots), hence are concerned with how to move them at run-time to optimise some desired metric (e.g. sampling accuracy). On the contrary, we assume static sensor devices that have been already deployed on a target area, without any prior knowledge of the actual spatial distribution of the phenomenon to observe. Finally, a great deal of related research contributions have the fundamental difference of adapting the sampling process to *domain-agnostic*, infrastructural properties such as residual energy, distance amongst devices, bandwidth consumption, rather than on the specific spatial distribution (and dynamics) of the phenomenon under monitoring [YF04, MR04, BC04, BC03, SGAP00, SAL+03, BC02, ÖFL04].

Narrowing down the research landscape just overviewed, we now describe and compare in more detail the approaches to spatial adaptive sampling most similar to ours, emphasising the most notable differences.

In reference [VS05] the Topology Adaptive Spatial Clustering algorithm is presented. It is a distributed algorithm that partitions a WSN into a set of disjoint sampling clusters, with no prior knowledge of cluster number or size (like ours), by encoding geographical distance, connectivity, and deployment density information in a single measure upon which leader election (for cluster heads) happens. The goal is to group together nodes in proximity and

within regions of similar deployment density, to improve efficiency of data aggregation and compression. Besides the focus on efficiency, that is only half the story in our approach (the other being accuracy, hence the trade-off), two are the fundamental differences our approach has with respect to this: first, in our case adaptation is domain dependent, in the sense that it depends on some property (e.g. variance) of the phenomenon under observation, not on infrastructural properties; second, we do also consider over-sampling when variation is high, whereas reference [VS05] only considers under-sampling where measures are redundant.

In reference [LXZ+13], another distributed approach to spatial adaptive sampling is presented. It is based on the assumption that neighbouring sampling nodes usually have similar readings (high spatial correlation), hence can be grouped in a cluster to improve energy consumption. The proposed algorithm uses such spatial correlation, two application-specific threshold parameters (error tolerance and correlation range), and residual energy to elect cluster heads, with the goal of minimising the number of clusters, and the variance of their size. The introduction of the two application-dependant parameters makes this proposal closer to ours, as they could be used to steer the adaptation toward domain-specific aspects, to some extent. However, most of the calculations still rely on infrastructural properties rather than measures about the phenomenon of interest, and the focus is, once again, on energy saving, thus, authors do not consider over-sampling but solely under-sampling.

Reference [LM07] shares with us the interest in performing adaptation based on the information observed by sensors, rather than on network, energy, or other infrastructural aspects. However, they consider a special case where the clusters must correspond to pre-determined "sources of interest", such as physical objects/devices in the environment. Moreover, clusters are formed with the secondary objective of being balanced, whereas we allow them to be of different sizes and shapes (and even encourage to be so depending on the spatial distribution of the phenomenon of interest).

Reference [LVP11] proposes SILENCE, a distributed, space-time adaptive sampling algorithm based on (space-time) correlation of measured values. The foremost goal pursued is efficiency: SILENCE in fact strives to minimise communication and processing overhead, by minimising sampling redundancy and also adapting (i.e. slowing down) the scheduling speed of sensor devices. However, yet again the basic assumption, and focus of the approach, is the case where spatial correlation of sampled values is high; whereas we explicitly focus on the opposite situation (while still providing a working solution even in the case considered by SILENCE). Furthermore, also SILENCE only considers down-sampling.

Finally, the ASample algorithm [SKS10] is the one approach most similar to ours, not in the techniques exploited, but in pursuing domain-driven adaptation, in doing so in a fully-distributed way, and in considering the opportunity to over-sample, too. In particular, ASample builds a Voronoi tessellation of the area where the WSN is deployed, in a fully distributed way by considering only neighbourhood information, instead of the whole topology. Such a tessellation considers a desired sampling accuracy, specified at the application level: while a given Voronoi region is within the accuracy bound, it keeps expanding; on the contrary, whenever the accuracy constraint is violated, a virtual centroid of a novel Voronoi region is spawned, with a value that is obtained through interpolation of the neighbouring regions. This is an important aspect to consider, as it introduces synthetic data, which is something we avoid as we only increase sampling granularity if there are actual devices available in the target area. Moreover, there is an assumption underlying the ASample approach that does not hold in general, and, specifically, it is in contrast with our intended goal (obtaining potentially irregular clusters to reflect the irregular spatial distribution of

the observed phenomenon): it is assumed that the smaller the area covered by the Voronoi region, the less representative the samples drawn are, hence the smaller the impact on the global sampling accuracy. In our targeted scenarios, the opposite could be true, too: smaller regions represent sharp variance of measurements across space, and more accurately represent the irregularities of the underlying phenomenon.

3. Distributed Aggregate Sampling: Model

In order to define the problem and characterise our approach, we leverage the event structure framework [NPW81, ABDV18], which provides a general model of situated computations. Within this formal framework, in this section we describe the computational model (Section 3.1), self-stabilisation as a desired property of solutions in this model (Section 3.2), and the spatial sampling problem that we tackle in this manuscript (Section 3.3). The computational model introduces the necessary terminology to understand both the problem formulation and the solution we propose in Section 4. Introducing the self-stabilisation property is required to be able to evaluate such a solution in its effort to adapt to the dynamics of the phenomenon of interest, both formally as done in Section 4.3 and practically as done later in Section 5.

3.1. Computational Model.

We consider a computational model where a set of *devices* (typically comprising *sensors* and *actuators* to perceive and act upon the environment) compute at discrete steps called *computation rounds* and interact with *neighbour devices* by exchanging messages. Executions of such systems can be modelled through *event structures* [NPW81, Pra86] as in [AVD+19, ABDV18][1]. Following the general approach in [ABDV18], we enrich the event structure with information about the devices where events occur.

Definition 3.1 (Situated event structure)**.** A *situated event structure* (ES) is a triplet $\langle E, \rightsquigarrow, d\rangle$ where:

- E is a countable set of ***events***;
- $\rightsquigarrow \subseteq E \times E$ is a ***messaging*** relation from a ***sender*** event to a ***receiver*** event (these are also called ***neighbour* events**);
- $d : E \to \Delta$, where Δ is a finite set of device identifiers δ_i, maps an event $\epsilon \in E$ to the device $d(\epsilon) \in \Delta$ where the event takes place.

The elements of the triplet are such that:

- the transitive closure of $\rightsquigarrow$ forms an irreflexive partial order $< \subseteq E \times E$, called ***causality*** relation (an event ϵ is in the ***past*** of another event ϵ' if $\epsilon < \epsilon'$, in the ***future*** if $\epsilon' < \epsilon$, or ***concurrent*** otherwise);
- for any $\delta \in \Delta$, the projection of the ES to the set of events $E_\delta = \{\epsilon \in E \mid d(\epsilon) = \delta\}$ forms a well-order, i.e., a sequence $\epsilon_0 \rightsquigarrow \epsilon_1 \rightsquigarrow \epsilon_2 \rightsquigarrow \dots$.

Additionally, we introduce the following notation:

- $recvs(\epsilon) = \{d(\epsilon') \mid \epsilon \rightsquigarrow \epsilon'\}$ to denote the set of receivers of ϵ, i.e., the devices receiving a message from ϵ;

[1] Our notion of an event structure does not use the conflict relation [NPW81], which is used to express non-determinism. Indeed, we only use partial ordering. Though it could be called a pomset [Pra86], we use this terminology to conform to previous research [ABDV18, VBD+19]. In our model, non-determinism is provided by the environment: then, a single event structure is used to describe *one* possible system execution, and results referring to multiple system executions universally quantify over all the possible event structures.

- $T^-_{\epsilon_0} = \{\epsilon : \epsilon < \epsilon_0\}$ to denote the past event cone of ϵ_0 (which is a finite set, since we assume the system has a starting point in time at any device);
- $T^+_{\epsilon_0} = \{\epsilon : \epsilon_0 < \epsilon\}$ to denote the future event cone of ϵ_0;
- $X|_{E'}$ to denote the projection of a set, function, or ES X to the set of events $E' \subseteq E$. Note that the projection of an event structure to the future event cone of an event is still a well-formed ES.

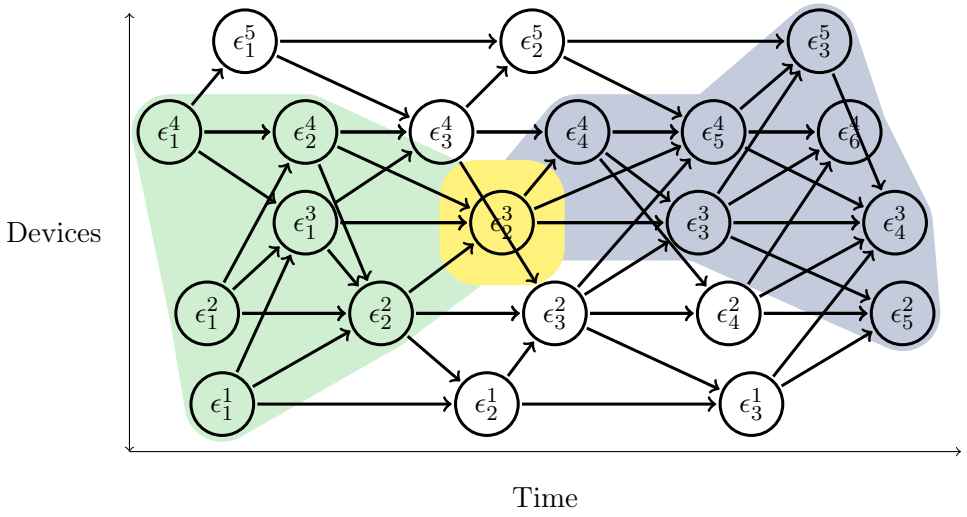


FIGURE 2. Example of an event structure. In the node labels, superscripts denote device identifiers, while subscripts are progressive numbers denoting subsequent rounds at the same device. The blue (resp. green) background denotes the future (resp. past) of a reference event denoted with a yellow background.

An example of ES is given in Figure 2, events denote computation rounds. Notice that self-messages (i.e., messages from an event to the next on the same device) can be used to model persistence of *state* over time. Also, notice that two subsequent events at some device (e.g., ϵ^4_4 and ϵ^4_5 of device 4 in Figure 2) may share a same sender event from a neighbour (ϵ^3_2 in this case): in a real system, this could be due to two distinct communication acts with the same message, as well as to a mechanism by which the receiving device reuses the most recently received message from that neighbour through some *message retention* policy (which is an implementation mechanism useful to support stability of neighbourhoods).

Remark 3.2 (Communication and distributed execution)**.** The description of a system execution as an event structure as per Definition 3.1 abstracts from details regarding the actual communication and scheduling mechanisms used in a deployed system. Concrete communication mechanisms may include point-to-point network channels, based on wired or wireless technologies, broadcasts, intermediaries (like, e.g., the cloud), or even stigmergic (i.e., environment-mediated) means. The computation rounds may be scheduled at a fixed frequency, using a particular time distribution, or reactively (e.g., in response to new sensor values or reception of messages from neighbours). A more in-depth discussion of how aggregate computing systems may be deployed and executed can be found in [CPP+20, PCV+21].

In the computation model we consider, based on [ABDV18], each event ϵ represents the execution of a program taking all incoming messages, and producing an outgoing message (sent to all neighbours) and a result value associated with ϵ. Such "map" of result values across all events defines a computational field, as follows.

Definition 3.3 (Computational field)**.** Let $\mathbf{E} = \langle E, \rightsquigarrow, d \rangle$ be an event structure. A *computational field* on $\mathbf{E}$ is a function $f : E \to \mathbb{V}$ that maps every event ϵ in E (also called the *domain* of the field) to some value in a value set $\mathbb{V}$.

Computational fields are essentially the "distributed values" which our model deals with; hence computation is captured by the following definition.

Definition 3.4 (Field computation)**.** Let $\mathbf{E} = \langle E, \rightsquigarrow, d \rangle$ be an event structure, and denote $\mathbb{F}_{E,\mathbb{V}}$ the set of fields on domain E and co-domain $\mathbb{V}$, i.e., $\mathbb{F}_{E,\mathbb{V}} = \{f : E \to \mathbb{V}\}$. Given two sets of values $\mathbb{V}, \mathbb{V}'$, a *field computation* over $\mathbf{E}$ is a function $\Phi_{\mathbf{E}} : \mathbb{F}_{E,\mathbb{V}} \to \mathbb{F}_{E,\mathbb{V}'}$ mapping an input field to an output field on the same domain of $\mathbf{E}$ (but possibly on a different co-domain).

This definition naturally extends to the case of zero or multiple input fields.

Now, we define the notion of a *field-based program*, which we denote as a construct expressing a computation on any possible *environment*, where an environment can be modelled by an event structure and fields over it denoting environmental values perceivable by devices (e.g., temperature fields would assign a temperature value to each event).

Definition 3.5 (Field-based operator)**.** A *field-based operator* (or *field-based program*) is a function P taking an event structure as input and yielding the field computation that would occur on it, namely $P(\mathbf{E}) = \Phi_{\mathbf{E}}$.

In other words, a field-based operator works as a "program" (design-time): it applies to a certain event structure to generate a resulting field computation (run-time). It would also be correct to say that a field-based program provides an implementation of a field-based operator, similarly to how, e.g., *quick-sort* provides an implementation of a sorting operator on lists. Notice also that we restrict our analysis to *computable* field-based operators as per previous work [ABDV18].

There exist core languages and full-fledged programming languages for conveniently expressing field-based programs: these are known as *field calculi* and *aggregate programming languages*, respectively [VBD$^+$19]. One of these is used in Section 4.2 to implement our adaptive spatial sampling algorithm. However, as the following example demonstrates, field-based programs can also be expressed, in our framework of event structures, by a *local* perspective, in terms of how inputs and ingoing messages at one event are mapped to an output and outgoing messages.

Example 3.6 (Gradient field computation and operator)**.** Term *gradient* commonly refers to a kind of distributed data structures for estimating the distance from any device in a network to its closest *source* device, and a family of distributed algorithms for building them [ACDV17], which are very useful for implementing self-organising systems [Nag02, VAB$^+$18, WH07].

In our framework, a distributed algorithm for building gradients (*gradient operator*) can be modelled as a field-based operator P_G mapping any event structure $\mathbf{E} = \langle E, \rightsquigarrow, d \rangle$ to a gradient field computation Φ_G over it. A *gradient field computation* $\Phi_G : \mathbb{F}_{\mathbf{E},Bool \times Metric} \to \mathbb{F}_{\mathbf{E},\mathbb{R}}$ is essentially a map:

- from an input field $f_i : \mathbb{F}_{\mathbf{E},Bool\times Metric}$ mapping each event $\epsilon \in E$ with a pair $(source, metric) \in Bool \times Metric$, where $Metric \subseteq E \times \mathbb{R}$, denoting *source* events/devices (those where $source = \top$), distinguished from non-source events/devices (those where $source = \bot$), and a metric associating neighbouring events to an estimation of the corresponding spatial distance, $metric(\epsilon) = \{(\epsilon', x) \mid \epsilon' \rightsquigarrow \epsilon \wedge x \in \mathbb{R}\}$;
- to the output field stabilising (cf. Section 3.2) to the minimum distances to source devices.

A simple operator for a gradient computation could be implemented through the following function, local to an event $\epsilon_0 \in E$ receiving from a possibly empty set of sender events ϵ_i (with ϵ_0 denoting the sender event at the same device, if any, and with $\epsilon_{i>0}$ the sender events from other devices), of the input field's value and the message set $\mathcal{M} = \{\epsilon_i \mapsto g_i\}$ providing the neighbours' current gradient estimates:

$$g(source, \{\epsilon_i \mapsto dist_i\}, \{\epsilon_i \mapsto g_i\}) = \begin{cases} 0 & source = \top \\ \min_{i>0}\{g_i + dist_i\} & source = \bot \wedge \{\epsilon_{i>0}\} \neq \emptyset \\ +\infty & \text{otherwise} \end{cases}$$

An example of the induced computation is shown in Figure 3, assuming a simple metric where $dist_0 = 0$ and $dist_{i>0} = 1$.

3.2. **Self-stabilisation.** We now provide the definitions necessary to model *self-stabilisation* following the approach in [VAB+18]. Namely, the following definitions capture the idea of *adaptiveness* whereby as the environment of computation stabilises, then the result of computation stabilises too, and such a result does not depend on previous transitory changes.

Definition 3.7 (Static environment)**.** An event structure $\mathbf{E} = \langle E, \rightsquigarrow, d\rangle$ is said to be a *static environment* if it has stable topology, namely all events of a given device always share the same set of receivers, i.e., $\forall \epsilon, \epsilon', d(\epsilon) = d(\epsilon') \Rightarrow recvs(\epsilon) = recvs(\epsilon')$.

Note that, following the approach in [VAB+18], we introduce the notion of a static environment to capture the eventual situation in which the environment stops perturbing the system. This is instrumental to rely on an abstract characterisation of self-stabilising computations, which are those in which the system keeps intercepting changes in the environment and adapting to them: whenever (and if) the environment becomes static, one can observe the result of that adaptation that eventually establishes.

Definition 3.8 (Stabilising environment)**.** An event structure $\mathbf{E} = \langle E, \rightsquigarrow, d\rangle$ is said to be a *stabilising environment* if it is eventually static, i.e., $\exists \epsilon_0 \in E$ such that $\mathbf{E}|_{T^+_{\epsilon_0}} = \langle E|_{T^+_{\epsilon_0}}, \rightsquigarrow|_{T^+_{\epsilon_0}}, d|_{T^+_{\epsilon_0}}\rangle$ is static. In this case we say it is *static since* event ϵ_0.

Definition 3.9 (Stabilising field)**.** Let event structure $\mathbf{E} = \langle E, \rightsquigarrow, d\rangle$ be a stabilising environment, static since event ϵ_0. A field $f : E \to \mathbb{V}$ is said *stabilising* if it eventually provides stable output (an output that does not change since some round), i.e., $\exists \epsilon > \epsilon_0$ such that $\forall \epsilon' > \epsilon, \epsilon'' > \epsilon$ it holds that $d(\epsilon') = d(\epsilon'') \implies f(\epsilon') = f(\epsilon'')$.

An example of a stabilising field, which can also be thought as being generated by a gradient computation (cf. Example 3.6), is provided in Figure 3. The environment is static since event ϵ^3_1 (every event in the future event cone of ϵ^3_1 has the same set of receivers), and from event ϵ^3_2 (excluded) it holds that each device does not change the value it produces in its rounds.

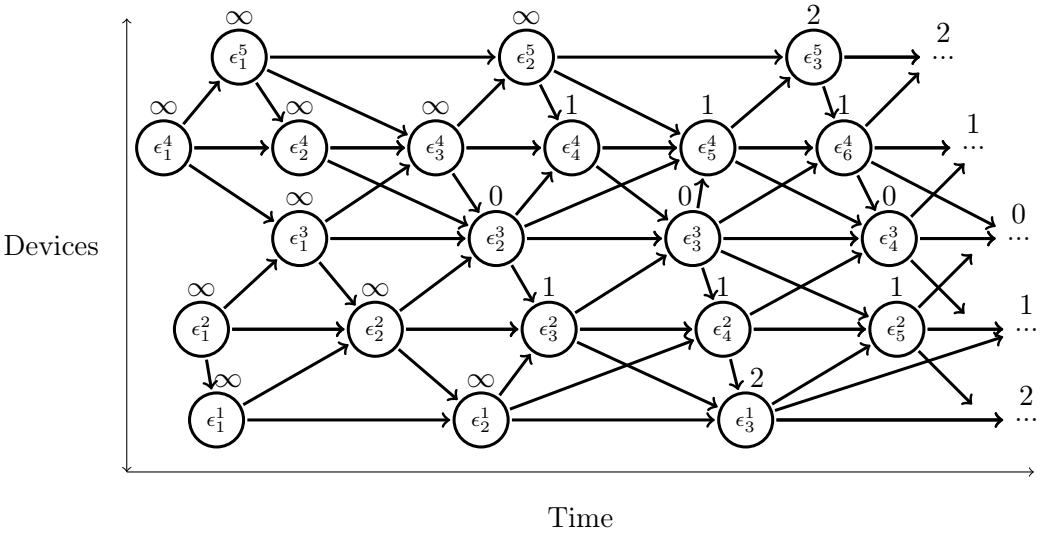


FIGURE 3. Example of a stabilising field. We use labels above the nodes to denote the values computed in the corresponding events, assuming the program is a gradient operator as per Example 3.6.

Definition 3.10 (Stabilising computation)**.** A field computation $\Phi_{\mathbf{E}} = \mathbb{F}_{E,\mathbb{V}} \to \mathbb{F}_{E,\mathbb{V}}$ is said *stabilising* if, when applied to a stabilising input field, it yields a stabilising output field.

Definition 3.11 (Self-stabilising operator)**.** A field-based operator (or program) P is said self-stabilising, if in any stabilising environment $\mathbf{E}$ it yields a stabilising computation $\Phi_{\mathbf{E}}$ such that, for any pair of input fields f_1, f_2 eventually equal, i.e. $f_1|_{T^+_\epsilon} = f_2|_{T^+_\epsilon}$ for some event ϵ, their output is eventually equal too, i.e., there exists a $\epsilon' > \epsilon$ such that $\Phi_{\mathbf{E}}(f_1)|_{T^+_{\epsilon'}} = \Phi_{\mathbf{E}}(f_2)|_{T^+_{\epsilon'}}$

Notice that universally quantifying over event structures, i.e., considering infinitely many system executions, makes finding decision procedures for properties like stabilisation undecidable *in general*. However, this does not prevent us from reasoning about such properties for a specific program, as we carry on in this paper—and as developed in previous works, e.g., in [VAB+18]. [2]

3.3. **Problem Definition.** We start by introducing the notion of *regional partition*, which is a finite set of non-overlapping contiguous clusters of devices: a notion that prepares the ground to that of an *aggregate sampling* which we introduce in this paper.

Definition 3.12 (Regional partition field, contiguous regions)**.** Let $\mathbf{E} = \langle E, \rightsquigarrow, d\rangle$ be a stabilising environment static since event ϵ_0. A *regional partition field* is a stabilising field $f : E \to \mathbb{V}$ on $\mathbf{E}$ such that:

- **(finiteness)** the image $Img(f) = \{f(x) \mid x \in E\}$ is a finite set of values;
- **(eventual contiguity)** there exists an event $\epsilon'_0 > \epsilon_0$ such that for any pair of events $\epsilon_1, \epsilon_n \in T^+_{\epsilon'_0}$, $f(\epsilon_1) = f(\epsilon_n)$ implies that there is a sequence of events $\epsilon_1 < ... < \epsilon_n$ connecting ϵ_1 to ϵ_n where $f(\epsilon_i) = f(\epsilon_1) = f(\epsilon_n)\ \forall 1 \leq i \leq n$.

[2] On the other hand, note that most of our definitions could be given considering finite runs, where proving decidability could be easier—but this is not developed for the sake of generality.

Note that the set of domains of regions induced by f is defined by $regions(f) = \{f^{-1}(v) : v \in Img(f)\}$. Moreover, given two regions $E, E' \in regions(f)$, we say that they are *contiguous* if $\exists \epsilon \in E, \epsilon' \in E' : \epsilon \rightsquigarrow \epsilon' \vee \epsilon' \rightsquigarrow \epsilon$.

An example of a regional partition field is shown in Figure 4. Notice that for any pair of events in the same space-time region there exists a path of events entirely contained in that region. Also, notice that, by this definition, different disjoint regions denoted by the same value r are not possible.

Definition 3.13 (Aggregate sampling)**.** An *aggregate sampling* is a stabilising computation $\Phi_S : \mathbb{F}_{E,\mathbb{V}} \to \mathbb{F}_{E,\mathbb{V}}$ that, given an input field to be sampled, yields as output a regional partition field.

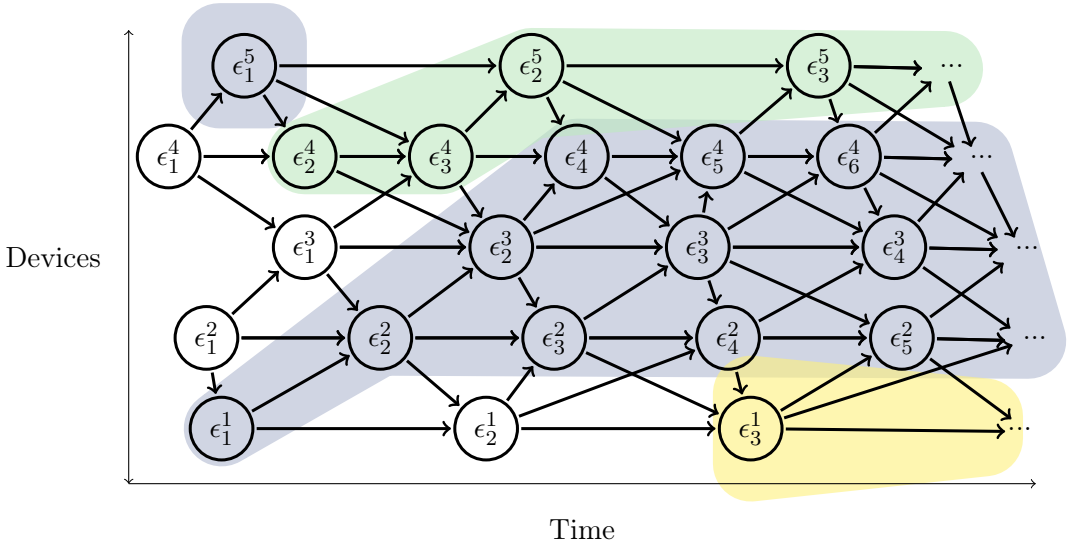


FIGURE 4. Example of a regional partition field with regions r_{blue}, r_{green}, r_{yellow}, r_{white} (the background is used to denote the output of the field). Notice that contiguity does not hold everywhere and anytime but only since event ϵ_2^3.

Once we have defined an aggregate sampling process in terms of its inputs, outputs, and stabilising dynamics, we need a way to measure the error introduced by the aggregate sampling. To this purpose, we introduce the notion of a *stable snapshot*, namely a field consisting of a sample of one event per device from the stable portion of a stabilising field.

Definition 3.14 (Stable snapshot)**.** Let $\mathbf{E} = \langle E, \rightsquigarrow, d \rangle$ be an event structure, and $f : E \to \mathbb{V}$ be a stabilising field on $\mathbf{E}$ which provides stable output from $\epsilon_0 \in E$. We define a *stable snapshot* of field f as a field obtained by restricting f to a subset of events in the future event cone of ϵ_0 *and* with exactly one event per device, i.e., a field $f_S : E_S \to \mathbb{V}$ such that $E_S \subseteq T^+_{\epsilon_0}$, *and* $\forall \epsilon, \epsilon' \in E_S : d(\epsilon) = d(\epsilon') \implies \epsilon = \epsilon'$, *and* $\forall \epsilon \in T^+_{\epsilon_0}, \exists \epsilon' \in E_S : d(\epsilon') = d(\epsilon)$.

Definition 3.15 (Stable snapshot error-distance)**.** We call *stable snapshot error-distance* any metric $\mu : \mathbb{F}_{E,\mathbb{V}} \times \mathbb{F}_{E,\mathbb{V}} \to \mathbb{R}_0^+$ over stable snapshots that feature same domain (event structure) and codomain (set of values).

We are now able to characterise adequacy properties for a sampling operator, intuitively capturing the fact that sampling correctly trades-off the size of regions with their accuracy. We first start by introducing a notion that handles accuracy, stating that any of the produced regions won't cause the error-distance to be over a certain threshold.

Definition 3.16 (Aggregate sampling error)**.** Let $\Phi_{\mathbf{E}} : \mathbb{F}_{E,\mathbb{V}} \to \mathbb{F}_{E,\mathbb{V}}$ be an aggregate sampling, and consider an input field $f_i : E \to \mathbb{V}$ and corresponding output regional partition $f_o : E \to \mathbb{V}$. We say that f_o *samples* f_i *within error* η *according to error-distance* μ, if the error-distance of stable snapshots of f_i and f_o in any region is not bigger than η, that is: let f_i^s and f_o^s be stable snapshots of f_i and f_o, then for any region $E' \in regions(f_o^s)$, we have $\mu(f_i^s|_{E'}, f_o^s|_{E'}) \leq \eta$.

Note that *accuracy* can be generally achieved simply by partitions defining many small regions—up to the corner case in which all regions include just one device, hence trivially induce zero error-distance. Therefore, we are also interested in *efficiency*, namely the ability of a regional partition to rely on as few regions as possible. Without a centralised approach, however, partitioning is necessarily sub-optimal, since it can rely only on local interaction/competition among regions, hence it should be expected that some regions will stop "expanding" as they reach a smaller threshold. Additionally, there can also be corner cases where regions with very small error-distance are created, e.g., because what remains to be covered in an iterative selection of regions is simply a very small part of the network, or one with rather uniform values introducing little sampling errors. What we may require from an adequate sampling operator, however, is that such regions are somewhat not the norm. This is formally captured by the following definition, essentially introducing a "lower bound" for the error-distance of regions.

Definition 3.17 (Local optimality of a regional partition)**.** Let $\Phi_{\mathbf{E}} : \mathbb{F}_{E,\mathbb{V}} \to \mathbb{F}_{E,\mathbb{V}}$ be an aggregate sampling, consider an input field f_i and corresponding output regional partition f_o such that f_o samples f_i within error η according to distance μ, and denote with f_i^s and f_o^s the stable snapshots of f_i and f_o, respectively (cf. Definition 3.16). We say that f_o is *locally optimal* under error η and with efficiency k ($k > 0$) if all pairs of contiguous regions $E', E'' \in regions(f_o)$ are such that $\mu(f_i^s|_{E' \cup E''}, f_o^s|_{E' \cup E''}) \geq k \cdot \eta$.

For example, we will show that the algorithm we propose guarantees $k = 0.5$ (see Section 4.3). Note that we call this notion "local optimality" to stress the fact that an identified partition is not necessarily the best one that could be found, but it is one that cannot be significantly improved with a small change, such as combining two regions—a small improvement is possible, depending on the efficiency factor k. This notion well fits our goal of dealing with dynamic phenomena and large-scale environments, where one is more geared towards finding good heuristics for self-organising behaviour.

So, we are now ready to define the goal operator for this paper.

Definition 3.18 (Effective sampling operator)**.** An *effective sampling operator* with efficiency k is a self-stabilising operator P_η, parametric in the error bound η, such that in any stabilising environment $\mathbf{E}$ and stabilising input f_i, a locally optimal regional partition with efficiency k and within error η is produced.

4. Aggregate Computing-based Solution

In this section, we define a *space-based adaptive sampling* algorithm, called AggregateSampler, (Section 4.1), discuss its implementation in *aggregate computing* [BPV15, VBD+19] (Section 4.2), and prove the algorithm is a self-stabilising, effective sampler with efficiency at least $k = 0.5$ (Section 4.3). The algorithm is defined in terms of the computational model described in Section 3.1, as well as its implementation, and is the one evaluated in Section 5. The proofs are based on the definitions of Section 3.2 and Section 3.3.

4.1. AggregateSampler Algorithm for Adaptive Spatial Sampling.

The problem of creating partitions in a self-organising way is very much related to a problem of multi-leader election [PCVN21, PCV22].

Building on this idea, our approach starts by solving a *sparse leaders election problem* [Lyn96], for which self-stabilising solutions exist [MADB20, BCC+21, PCV22]. Leaders are used as *samplers* of the input field. During the election of leaders/samplers, we associate them with larger and larger regions of "follower devices" that will provide the sampled value as output. During execution of the algorithm, such regions will expand until the desired error-distance can be kept under the threshold η. This process is managed so that there won't be any overlap with other regions, and so that no devices of the network remain outside of some region (i.e., each device will follow exactly one leader).

To ensure that regions are connected, and won't overcome the threshold independently of the chosen leader, we adopt as error-distance one based on "distance among devices", as follows. The algorithm can be configured to adopt any strategy that is able to turn input and output fields into a metric m for devices: such a metric is as usual a function mapping a pair of neighbour devices to a non-negative real number, called the "local sampling distance" of the two devices—intuitively, the higher the physical distance of devices and the higher the difference of input values and output values of the two devices, the higher is m for that pair. Given this metric, any pair of devices of the network can be associated with a *path sampling error*, which is the size of the shortest path (according to the metric) connecting the two devices. The proposed algorithm will then produce regions adopting as error-distance μ the maximum path sampling error of any pair of devices in the region, and it will turn out that any pair of contiguous regions combined will necessarily give error-distance greater than $0.5 * \eta$ (efficiency $k = 0.5$).

The algorithm is defined as follows (see Figure 5):

(1) each device announces its candidature for leadership;
(2) each device propagates to its neighbours the candidature of the device it currently recognises as leader, its sampled value, and the path sampling error from it, fostering the expansion of its corresponding region;
(3) devices discard candidatures whose path sampling error from the leader exceeds half the expected threshold ($\eta/2$);
(4) in case multiple valid candidatures (i.e., those that are not discarded) reach a device, one is selected based on a *competition policy*.

The specific strategies for computing the local sampling distance and the leader competition policy are application-dependent—we will provide some instances in Section 5.

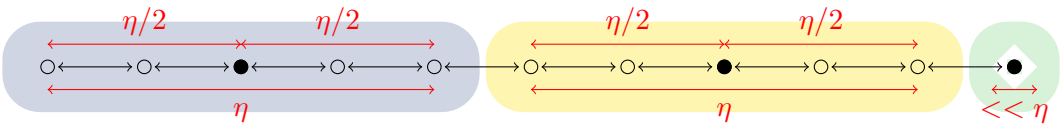


FIGURE 5. Example of a regional partitioning (with three contiguous regions) created by the algorithm on a simplified system where devices are arranged on a line. Notation: black dots denote the leaders/samplers; the coloured areas denote regions; and the red extension lines are used to denote the error-distances. Note that no device in a region can have path sampling error greater than $\eta/2$ with respect to the leader, and that very small regions can still exist in corner cases (e.g., the green region on the right).

Competition and leader strength. Although competition among leaders could be realised in several ways, many techniques may lead to non-self-stabilising behaviour: for instance, if the winning leader is selected randomly in the set of those whose error is under threshold, regions may keep changing even in a static environment. In this work, we propose a simple strategy: every leader associates its candidature with the local value of a field that we call *leader strength*; in case of competing candidatures, the highest such value is selected as winner, breaking the symmetry. The leader strength can be of any orderable type, and its choice impacts the overall selection of the regions by imposing a selection priority over leaders (hence on region-generation points). If two candidate leaders have the same strength, then we prefer the closest one. If we are in the (unlikely) situation of perfect symmetry, with two equally-strong candidate leaders at the same distance, then their device identifier is used to break symmetry.

Region expansion and path sampling error. Inspired by previous work on distributed systems whose computation is independent of device distribution [BVPD17], the proposed approach essentially accumulates the path sampling error along the path from the leader device towards other devices along a gradient (cf. Example 3.6), a distributed data structure that can be generated through self-stabilising computations [VAB+18] (cf. Section 3.2). We thus have two major drivers:

(1) the leader strength affects the creation of regions by influencing the positions of their source points;
(2) the path sampling error influences the expansion in space of the region across all directions, mandating its size and (along with the interaction with other regions) its shape.

For instance, a metric could be the absolute value of the difference in the perceived signal (e.g., a value sampled from a sensor—cf. Section 3.1) between two devices: devices perceiving very different values would tend not to cluster together (even if spatially close), as they would perceive each other as farther away (leading to irregular shapes).

As the simulations in the next section verify, connecting region expansion with the error-distance (i.e., using the error-distance as a distance metric for gradient computation) enables the determination of locally optimal sampling regions. We recall that the local optimality property means that all regions are essentially needed except for corner cases.

```
1  // Definition of the record (product type) Sample + accessor functions
2  def Sample(symmetryBreaker, distance, leaderId) = [symmetryBreaker, distance, leaderId]
3  def breakSymmetry(sample) = sample.get(0)
4  def sampleDistance(sample) = sample.get(1)
5  def areaCenter(sample) = sample.get(2)
6  def discard() = Sample(POSITIVE_INFINITY, POSITIVE_INFINITY, POSITIVE_INFINITY)
7  // Logic to control the propagation of candidacies
8  def expansionLogic(sample, localId, radius) =
9    mux (areaCenter(sample) == localId || sampleDistance(sample) >= radius) {
10     discard()
11   } else {
12     sample
13   }
14
15 def AggregateSampler(mid, radius, symmetryBreaker, metric) {
16   let local = Sample(-symmetryBreaker, 0, mid)
17   areaCenter(
18     share (received <- local) {
19       let candidacies = received.set(1, received.get(1) + metric())
20       let filtered = expansionLogic(candidacies, mid, radius)
21       min(local, foldMin(discard(), filtered))
22     }
23   )
24 }
```

FIGURE 6. Source code of the algorithm.

4.2. **Aggregate Computing-based Implementation.** An implementation of the algorithm expressed in the Protelis aggregate programming language [PVB15] is shown in Figure 6. In aggregate computing, a so-called *aggregate program* such as the one shown in Figure 6, is repeatedly run by all the devices: it expresses a logic for mapping the *local context* (given by sensor readings and messages from neighbours) to an *output value* and an *output message* to be sent to all the neighbours. In other words, aggregate computing leverages the computational model described in Section 3.1, where each event denotes a full execution of the aggregate program against the event's inputs, determining the message payload passed to receiving events.

The core of the program is function AggregateSampler, which consists of the following main elements:

- *sampler candidacies are modelled as ordered triplets*, using 3-element tuples, with corresponding accessor functions (Figure 6, Lines 1–6), of the elements:
  (1) `simmetryBreaker`: a value used to break symmetry, capturing the "strength" of a candidacy;
  (2) `distance`: a value capturing the distance to the sampler node of a candidacy (e.g., computed through a gradient—cf. Example 3.6);
  (3) `leaderId`: holding the device identifier of the candidate sampler;
- function `expansionLogic` (Figure 6, 8–13) is defined to determine when a candidacy has to be discarded, i.e., when it comes from the device itself or when the distance of the candidate sampler is greater or equal than a `radius` parameter;
- function `AggregateSampler` (Figure 6, 15–24) is the entry point of the algorithm, parametrised in terms of the executing device identifier (`mid`), a maximum spatial range of

candidacies (`radius`), a value to break symmetry (`symmetryBreaker`), and a `metric` function providing distances to neighbours;
- `share(x <- init){ e }` is a bidirectional communication construct [ABD+20], that works as follows: the declared variable `x`, which is set to `init` at the first round, collects the evaluations of the overall `share` expression in neighbour devices (including the device itself), and the new value for the current device (which is the data item that will be sent to neighbours) is obtained by evaluating expression `e`;
- *the distance field of the neighbour candidacies is updated*, (Figure 6, Line 19), by adding to each candidacy provided by a neighbour the local distance w.r.t. that neighbour[3] (as provided by `metric`);
- *neighbours' candidacies that are too far or support the current device are de-prioritised* (Figure 6, Lines 20), by function `expansionLogic()`; and, finally
- *selecting the winner over the processed candidacies through minimisation*, by `min` and `foldMin` (Figure 6, Line 21), which minimise over the filtered candidacy triplets, with default candidacy as the one with the lowest priority (provided by `discard()`), and against the `local` candidacy.

The above described algorithm is an effective sampling operator (Definition 3.18) as long as (i) *half the path sampling error η is used as parameter* `radius`, so that function `expansionLogic` does not expand regions beyond the given error η, and (ii) *an additive* `metric` *is used*, so that it is impossible to decrease the error by expanding any given region (at best, it will stay the same).

4.3. **Formal Analysis.** In this section, we prove that the proposed solution is self-stabilising (cf. Definition 3.10, Definition 3.11) and that it represents an effective sampling operator leading to a bounded-error locally optimal regional partition (cf. Definition 3.13, Definition 3.17, Definition 3.18).

Since our aggregate sampling must be a stabilising computation (see Definition 3.13), we start by proving that our algorithm is self-stabilising. We do so by exploiting the framework in [VAB+18, ABD+20], which defines a set of self-stabilising *fragments* which can be composed together to yield self-stabilising operators (Definition 3.11). In particular, in [VAB+18] it is proved that any closed expression in the self-stabilising fragment is self-stabilising, by structural induction on the syntax of expressions and programs ([VAB+18], Appendix E, Lemma 2): values and variables are already self-stabilised, a function application self-stabilises (by the inductive hypothesis) if its arguments are self-stabilising, and similar considerations can be done for the other program fragments.

Theorem 4.1 (AggregateSampler is self-stabilising)**.**

Proof. In [ABD+20], it is proved that an expression of the following form (called a *minimising share* pattern) is self-stabilising:

```
1 share(x <- e) { fR(minHoodLoc(fMP(x), e), x) }
```

where (see Section 5.2 in [VAB+18] and, especially, Section 4.7 and Figure 7 in [ABD+20]):
- `fR(x, prev)` is a "raising function", with respect to partial orders, of `x` and `prev` (the value of `x` at the previous round);

[3]Note that this operation, together with the `share` application, essentially provides the same structure as the basic gradient algorithm discussed in Example 3.6.

```
def updateDistance(x, metric) {
  x.set(1, x.get(1) + metric())
  x
}
def fR(x, prev) = x
def fMP(x, localId, radius, metric) =
  expansionLogic(updateDistance(x, metric), localId, radius)
def minHoodLoc(e, loc) = min(loc, foldMin(discard(), e))
def AggregateSampler(mid, radius, symmetryBreaker, metric) {
    let local = Sample(-symmetryBreaker, 0, mid)
    areaCenter(
        share (x <- local) {
            let candidacies = x.set(1, x.get(1) + metric())
            let filtered = expansionLogic(candidacies, mid, radius)
            min(local, foldMin(discard(), filtered))
        }
    )
}
```

(A) First step of the conversion to a minimising-share form: `received` has been renamed to `x`; functions `updateDistance`, `fR`, `fMP`, and `minHoodLoc` have been defined.

```
// ... omitted ...
    areaCenter(
        share (x <- local) {
            let filtered = fMP(x, mid, radius, metric)
            min(local, foldMin(discard(), filtered))
        }
    )
// ... omitted ...
```

(B) The update to the distance field and the call to `expansionLogic` have been replaced by `fMP`.

```
// ... omitted ...
    share (x <- local) {
        let filtered = fMP(x, mid, radius, metric)
        min(local, foldMin(discard(), filtered))
    }.get(2)
// ... omitted ...
```

(C) Replace `areaCenter` with its definition (selection of the second element in the tuple).

```
// ... omitted ...
    share (x <- local) {
        minHoodLoc(fMP(x, mid, radius, metric), local)
    }.get(2)
// ... omitted ...
```

(D) Replace the share body with a call to `minHoodLoc`.

FIGURE 7. Syntactic steps for passing from Figure 6 to Figure 8.

- `fMP` is a monotonic progressive function of `x`, which can take further arguments as far as they are self-stabilising expressions that do not contain the share-bounded variable `x`; and

```
1  def updateDistance(x, metric) {
2    x.set(1, x.get(1) + metric())
3    x
4  }
5  def fR(x, prev) = x // raising function
6  def fMP(x, localId, radius, metric) = // monotonic progr.
7    expansionLogic(updateDistance(x, metric), localId, radius)
8  def minHoodLoc(e, loc) = // minimum of loc and e's values
9    min(loc, foldMin(discard(), e))
10 def AggregateSampler(mid, radius, symmetryBreaker, metric) {
11   let local = Sample(-symmetryBreaker, 0, mid)
12   share (x <- local) {
13     fR(minHoodLoc(fMP(x, mid, radius, metric), local), x)
14   }.get(2)
15 }
```

FIGURE 8. Protelis code from Figure 6 rewritten to conform to the *minimising share* self-stabilising pattern as per the Proof of Theorem 4.1.

- `minHoodLoc(e, loc)` selects the minimum among the neighbours' values of expression `e` and the current device's local value `loc`.

Now, the block of Protelis code (Figure 6) in Lines 15–24 can be rewritten as shown in Figure 8 that conforms to the minimising `share` pattern, where:

- the raising function `fR` is an identity on the first parameter, which is a trivially valid raising function (see Example 5.5 in [VAB+18]);
- function `expansionLogic` is a valid monotonic progressive function `fMP` of `x`, since it transforms neighbours' candidacies supporting the current device and those at a distance farther than `radius` to the highest value for the data type (cf. `discard()`), leaving the others unaltered; none of the provided additional arguments (`id`, `radius`, and `metric`) contains the `share`-bounded variable `x`.

More gradually, the transformation can be obtained by:

(1) renaming `received` to `x`, and defining functions `fR`, `fMP`, and `minHoodLoc` (Figure 7a);
(2) realising that `fMP` is a valid replacement for the combination of distance field update and call to `expansionLogic` and replacing accordingly (Figure 7b);
(3) replacing `areaCenter` with its definition (Figure 7c);
(4) replacing the `share` body with `minHoodLoc`, as they perform the same operation (Figure 7d);
(5) adding a call to `fR`, which is an identity function, does not alter the behaviour of the code and leads directly to the code in Figure 8.

The other elements in the program are only operations on local data which are also self-stabilising expressions. Since the AggregateSampler function consists exclusively of self-stabilising expressions, it is in turn self-stabilising [VAB+18, ABD+20]. □

Theorem 4.2 (AggregateSampler is an effective sampling operator)**.**

Proof. To prove that our algorithm represents an effective sampling operator P_η = AggregateSampler, we have to prove that it yields, on any stabilising input field f_i, an output stabilising field f_o of locally optimal regional partitions. As per Theorem 4.1, P_η

is self-stabilising, so on a stable input it will yield a stable output: let f_o^s be its snapshot, and f_i^s the corresponding input.

On the one hand, accuracy is guaranteed since AggregateSampler ensures that no device has path sampling error greater than $\eta/2$ from the leader: for the triangular inequality property of metric spaces ($m(a,b) \leq m(a,c) + m(c,b)$) this ensures that stable snapshot distance μ does not overcome η.

On the other hand, for local optimality under error η and distance μ, there must not exist two contiguous regions $E', E'' \in \mathit{regions}(f_o^s)$, with samplers δ' and δ'', where $\mu(f_i^s|_{E' \cup E''}, f_o^s|_{E' \cup E''}) \leq \eta/2$. Suppose two such regions exists, and let δ' be stronger than δ'' (i.e., it has higher symmetry breaker). Then, the path sampling error between δ' and δ'' is necessarily higher than $\eta/2$, because of the steps 3 and 4 of the algorithm: in fact, if it were smaller than $\eta/2$ then δ'' would have followed δ', and would not have been a leader (cf. Figure 9). □

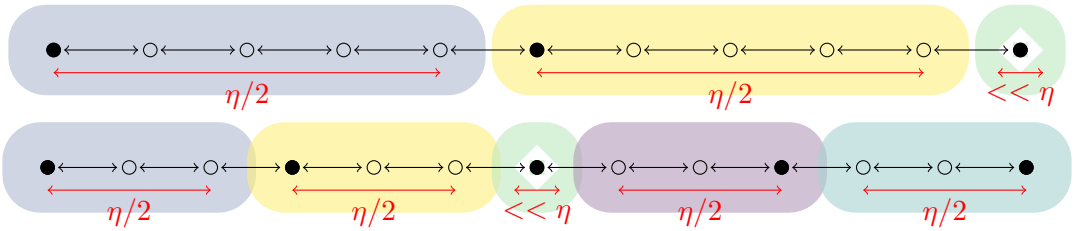


Figure 9. Examples of regional partitionings with efficiency 0.5. Notation: black dots denote the leaders; the coloured areas denote regions; and the red extension lines are used to denote the error-distances. Note how the union of the green singleton region (associated to the weakest leader) with its neighbouring regions would make the error-distance of the latter exceed $\eta/2$—if that would not be the case, then the former region would not have existed in the first place (cf. Theorem 4.2).

4.4. **Cost analysis.** Executing one cycle of the proposed algorithm requires operating over the information received from all the neighbours, which is, of course, proportional to their number. Operations within the `share` block (Lines 19–21 in Figure 6) are, indeed, operations on fields: by the aggregate computing semantics, they are evaluated for each neighbour. Thus, computationally, the cost of the algorithm is proportional to neighbourhood size: larger neighbourhoods require more effort.

From the point of view of message size, the payload of the algorithm has two components: the data type used to represent the sample, and, potentially, additional data that needs to be shared to compute the result of the `metric` function. The former depends on the actual types used for `symmetryBreaker` and `leaderId`, the latter on the type of distance returned by `metric`.

For example, assuming a classic TCP/IPv6 network and devices with a single network interface, we could use the local MAC address (6 bytes) as `symmetryBreaker`, the IP address (16 bytes) as `leaderId`, and a 4-byte floating-point number as return type of `metric`, resulting in a payload of 26 bytes per device per round. Additionally, however, further data might have to be shared to compute the `metric`; for instance, if devices are equipped with a GPS,

they may compute distances by sharing their coordinates and using the Haversine algorithm. Assuming a local sensor named `gps`, the Protelis code for such implementation of `metric` could be:

```
1 def distanceWithGps() {
2   let latLong = env.get("gps")
3   haversine(latLong, nbr(latLong))
4 }
```

The `nbr` call would incur into an additional network cost, as the local position would be shared with all neighbours. Assuming a couple of 4-bytes floating-point numbers for latitude and longitude, that would result into an additional 8 bytes per device per round, bringing the total up to 34 bytes from the initial 26 bytes. Notice, however, that this additional cost could get nullified if the `metric` function is implemented in a way that does not require additional data to be shared (for instance, by using the wireless signal strength as a proxy for the distance, or by using the hop distance).

If the algorithm is implemented in an aggregate computing language, and no application-specific optimisation is devised, an identifier for each interaction (`share` or `nbr` call) is attached to the message, so the payload would also include the size of one or two identifiers (again, depending on whether the `metric` requires data to be shared).

5. Evaluation

This section discusses the evaluation of the algorithm proposed in Section 4 against the properties defined in Section 3 by means of simulation. We first present the evaluation goals (Section 5.1), scenarios (Section 5.2), parameters (Section 5.3), evaluation metrics (Section 5.4), and main implementation details (Section 5.5), and finally provide a discussion of the results (Section 5.6). The whole experimental framework has been published as permanently available artefacts [Pia22, Pia23b] in Zenodo, with instructions for replicating the results.

5.1. **Evaluation Goals.** In this section, we validate the behaviour of the proposed effective aggregate sampling algorithm. The goals of the evaluation are the following:

- *stabilisation*: we expect the algorithm to be *self-stabilising* (as per Definition 3.11 and Theorem 4.1), and thus to behave in a self-stabilising way under different conditions;
- *high information (entropy)*: we expect the algorithm to split areas with different measurements, namely, to dynamically increase the number of regions on a per-need basis to minimise the *aggregate sampling error* (as per Definition 3.16);
- *error-controlled upscaling*: we expect the algorithm to not abuse of region creation, but to keep the minimum number of regions (hence of the largest possible size—efficiency) required to maintain accuracy (as per Definition 3.17 and Theorem 4.2), intuitively, grouping together devices with similar measurements.

Clearly, upscaling and high information density are at odds: maximum information is achieved by maximising the number of regions, and thus assigning each device a unique region; however, doing so would prevent any upscaling. On the other hand, the maximum possible upscaling would be achieved when all devices belong to the same region, thus minimising information. We want our regions to change in space "fluidly" and opportunistically tracking

the situation at hand, achieving a trade-off between upscaling and amount of information (as per Definition 3.18).

5.2. **Scenarios.** We challenge the proposed approach by letting the algorithm operate on synthetic and realistic scenarios.

In the synthetic scenarios, we use different deployments of one thousand devices and different data sources. We deploy devices into a square arena with different topologies:

i) *grid (regular grid)*: devices are regularly located in a grid;
ii) *pgrid (perturbed/irregular grid)*: starting from a grid, devices' positions are perturbed randomly on both axes;
iii) *uniform*: positions are generated with a uniform random distribution;
iv) *exp (exponential random)*: positions are generated with a uniform random distribution on one axis and with an exponential distribution on the other, thus challenging device-distribution sensitivity.

In all cases, we avoid network segmentation by forcing each device to communicate *at least* with the eight closest devices. We simulate the system when sampling the following phenomena:

i) *Constant*: the signal is the same across the space, we expect the system to upscale as much as possible;
ii) *Uniform*: the signal has maximum entropy, each point in space has a random value, we thus expect the system to create many small regions;
iii) *Bivariate Gaussian (gauss)*: the signal has higher value at the centre of the network, and lower towards the borders, producing a Gaussian curve whose expected value is located at the centre of the network, we expect regions to be smaller where the data changes more quickly;
iv) *Multiple bivariate Gaussian (multi-gauss)*: similar to the previous case, but the signal value is built by summing three bivariate Gaussian whose expected value is one third of the previous Gaussian, and whose expected values are located along the diagonal of the network (bottom-left corner, centre, top-right corner);
v) *Dynamic*: the system cycles across the previous states, we use this configuration to investigate whether and how the proposed solution adapts to changes in the structure of the signal.

In the realistic scenario, we use air quality data from the European Environment Agency [Pia23a], and, specifically, the PM10 data from February 2020 (included) to May 2020 (excluded). We position the sensor stations in their correct position as reported by the agency, and assume logical connectivity with close-by stations. We force each station to communicate with at least the closest stations, and we ensure that no network segmentation exists by enforcing full network reachability. This results in a much sparser network than the synthetic ones, and whose variance in the number of neighbours is much higher: some stations located in places far from geographical Europe (such as Réunion and other French overseas departments) have very few connections (possibly, a single one), while sensors located in dense urban areas can have dozens. To emulate energy-constrained devices, such as LoRaWAN motes, we limit the operating frequency of each device to $1/1800$Hz (namely, one round every half hour on average).

5.3. **Parameters.** The proposed solution can be tuned by three main parameters: the leader strength, the error tolerance, and the distance metric. In the experiments, we fix the error tolerance to a constant value, while we choose among three different alternatives for the leader strength and the distance metric.

For the leader strength parameter we use the local concentration of PM_{10} in the realistic experiment, while in the synthetic one we consider:

i) *value*: the local value of the tracked signal s;
ii) *mean*: the neighbourhood-mean value of the tracked signal s, assuming N to be the set of neighbours (including the local device), and s_i to be the value of the tracked signal at device $i \in N$, the value is computed as:

$$M = \frac{\sum_{i \in N} s_i}{|N|}$$

iii) *variance*: the neighbourhood-variance of the tracked signal s, assuming M_i to be the neighbourhood-mean computed at device $i \in N$, the value is computed as:

$$\frac{\sum_{i \in N} (M_i - s_i)^2}{|N|}$$

For the metric, in the synthetic scenario, we consider:

i) *distance*: the spatial distance is used as distance metric;
ii) *diff*: assuming that s_i is the value of the tracked signal at device i, the distance between two neighbouring devices a and b is measured as:

$$e_{ab} = e_{ba} = min(\epsilon, |s_a - s_b|)$$

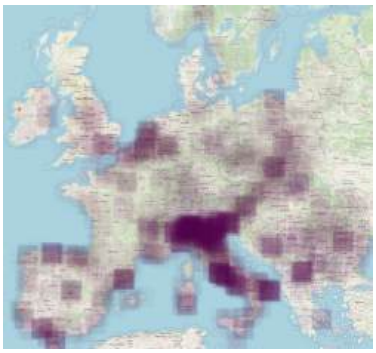
where $\epsilon \in \mathbb{R}_+$, $\epsilon = 0$ iff $a = b$, $0 < \epsilon \ll 1$ otherwise, we bound the minimum value to preserve the triangle inequality;
iii) *mix*: we mix the two previous metrics so that both the error and the physical distance affect in the distance definition; i.e., assuming $\overline{ab}$ to be the spatial distance between devices a and b, we measure the mix metric as:

$$\overline{ab} \cdot e_{ab}$$

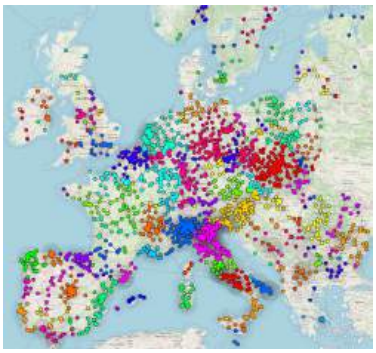
For the realistic scenarios we use instead:

i) *dist*: the spatial distance is used as distance metric;
ii) $dist_B$: same as *dist*, but country borders are considered as barriers;
iii) $\sigma(PM_{10})$: we weight the distance between neighbouring devices by a factor that depends on the Air Quality Index (AQI) value at the device location—devices with more different AQIs are considered more distant;
iv) $\sigma(PM_{10})_B$: same as $\sigma(PM_{10})$, but country borders are considered as barriers.

The idea is to challenge the algorithm by looking at how it behaves when operating on a network with a sparser and more heterogeneous structure, as well as to investigate the impact of arbitrary limits and non-linearities (e.g., the country borders) unrelated with the underlying signal included in the expansion metrics. Figure 10 shows a snapshot in time of the partitioning generated by the simulator in the realistic scenario for each of the aforementioned metrics.

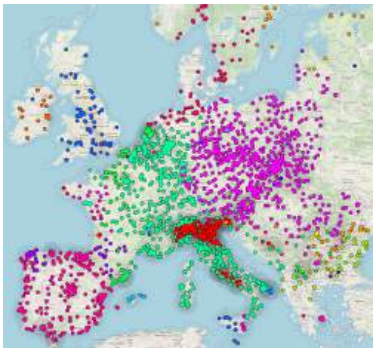


(A) PM_{10} concentrations (darker areas have higher PM_{10}).



(B) Partitions computed using $dist$.

(C) Partitions computed using $dist_B$.



(D) Partitions computed using $\sigma(PM_{10})$.

(E) Partitions computed using $\sigma(PM_{10})_B$.

FIGURE 10. Snapshots produced by the simulator at the same moment. Notice that the latter metrics ($\sigma(PM_{10})$ and $\sigma(PM_{10})_B$) are capable of capturing and managing contiguous areas with similar levels of air quality, tracking the underlying spatial structure of the signal.

5.4. **Evaluation Metrics.** We evaluate the system behaviour by considering the following evaluation metrics. Assume, at any time instant, that a set D of devices is partitioned into a set of regions $R = R_1 \cup \cdots \cup R_{|R|}$, where each R_r is a set of devices $\{D^r_1, \ldots, D^r_{|R_r|}\}$, and each device D^r_d senses the local value of the tracked signal s^r_d:

- *Region count $|R|$ (regions).* Counting the regions provides an indication about efficiency (Definition 3.17): more partitions should be expected in environments where the sampled signal has higher entropy.
- *Mean region size $\mu_R = \sum_{i=1}^{|R|} |R_i|/|R|$ (devices).* Related to the region count, but density-sensitive: when devices are distributed irregularly (as in the **exp** deployment, see Section 5.2), we expect this metric to be less predictable.
- *Standard deviation of the mean of the signal in regions $\sigma(\mu_s)$ (same unit of the signal).* This is a proxy for inter-region difference, with higher values denoting larger differences

between different regions. The mean signal inside region R_r is computed as:

$$\mu_s^{R_r} = \frac{\sum_{i=1}^{|R_r|} s_i^r}{|R_r|}$$

while the mean of the means of the signal is

$$\mu_s^R = \frac{\sum_{i=1}^{|R|} \mu_s^{R_i}}{|R|}$$

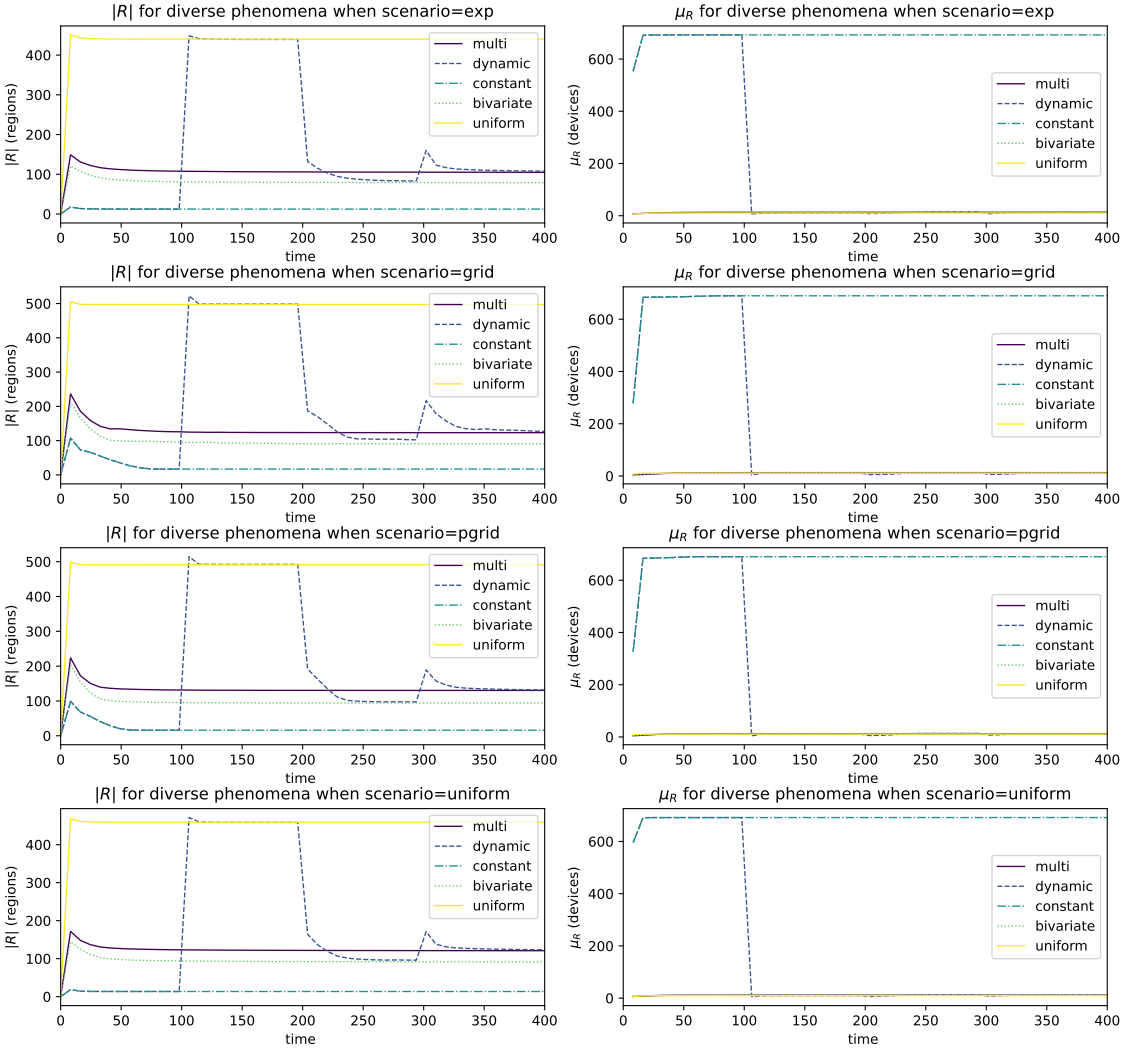


FIGURE 11. Region count (left column) and size (right column) across deployments and scenarios. The system behaves very similarly regardless of the device disposition. As expected, the higher information density leads to a larger number of smaller regions. The dynamic scenario shows that the partitions change in response to changes in the signal.

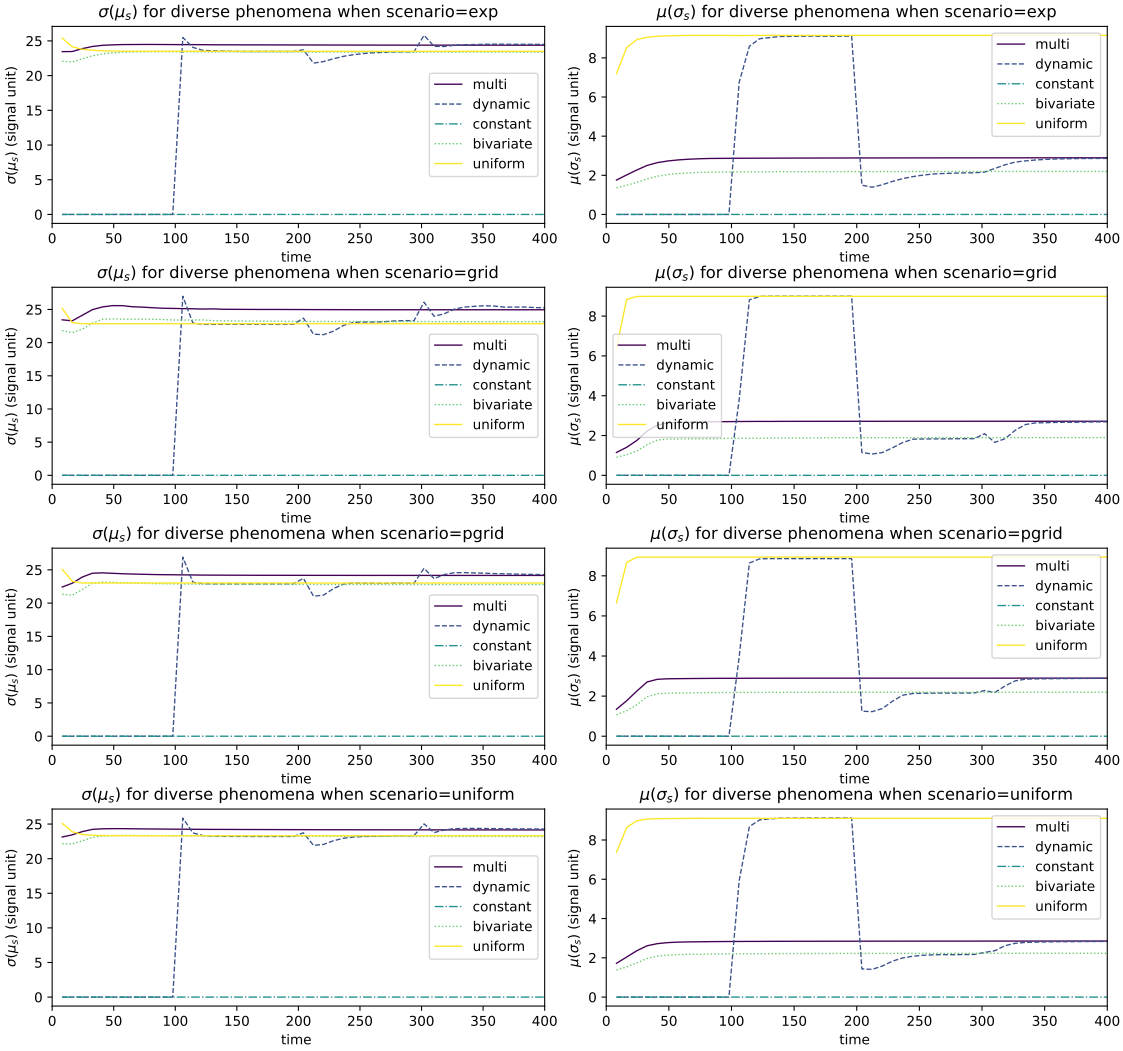


FIGURE 12. Standard deviation of the mean region value (left) and mean standard deviation (right) across deployments and scenarios, indicating respectively how much the regions readings differ from each other (the higher the more different) and how the regions are internally similar (the lower the more homogeneous are regions). The constant and uniform random signals work as baselines: in the former case, very large areas gets formed, while in the latter most regions count a single device (as expected). In the other cases, inter-region differences is maximised (they get as high as the most extreme case) keeping internal consistency under control.

thus

$$\sigma(\mu_s) = \sqrt{\frac{1}{|R|}\sum_{i=1}^{|R|}(\mu_s^{R_i} - \mu_s^R)^2}$$

- *Mean standard deviation of the signal in regions $\mu(\sigma_s)$ (same unit of the signal).* This is a proxy for the intra-region error. The lower this value, the more similar are the signal readings inside regions, hence the lower the error induced by the grouping (Definition 3.16). The standard deviation of the tracked signal inside region R_r is computed as:

$$\sigma_s^{R_r} = \sqrt{\frac{1}{|R_r|}\sum_{i=1}^{|R_r|}(s_i^r - \mu_s^{R_r})^2}$$

  thus

$$\mu(\sigma_s) = \frac{\sum_{i=1}^{|R|}\sigma_s^{R_i}}{|R|}$$

- *Standard deviation of the standard deviation of the signal in regions $\sigma(\sigma_s)$ (same unit of the signal).* Proxy metric for the consistency of partitioning. Higher values suggest that partitions have different internal error, hence behave differently (striving to satisfy Definition 3.18). It is computed as:

$$\sigma(\sigma_s) = \sqrt{\frac{1}{|R|}\sum_{i=1}^{|R|}(\sigma_s^{R_r} - \mu(\sigma_s))^2}$$

5.5. **Implementation and Reproducibility.** We rely on an implementation coded in the Protelis aggregate programming language [PVB15]. The simulations are implemented in the Alchemist simulator [PMV13]. The data analysis leverages Xarray [HH17] and matplotlib [Hun07].

In the synthetic scenarios, for each element in the Cartesian product of the device deployment type, signal form, leader strength, and distance metric, an experiment was carried out. Each experiment has been repeated 100 times with different random seeds, resulting in multiple simulation runs per experiment. Random seeds control both the evolution of the system (i.e., the order in which devices compute) and their position on the arena (except for the regular grid deployment, which is not randomised).

For the realistic scenario, since the position of the devices is mandated by the real-world deployment, we run 10 simulation repetitions for each experiment; in this case, the random seed controls the evolution of the system (the order in which the devices compute).

The presented results are obtained from taking the average of the metrics across all the repetitions; when a chart does not mention some parameters, then the results that are presented are also averaged across all values the parameter may assume for the simulation set. The experiment has been open sourced, publicly released[4][5], documented, equipped with a continuous integration system to guarantee replicability, and published as a permanently available, reusable artefact [Pia22].

[4] `https://github.com/DanySK/Experiment-2022-Coordination-Space-Fluid`

[5] `https://github.com/DanySK/experiment-2023-lmcs-pm10-pollution-space-sampling`

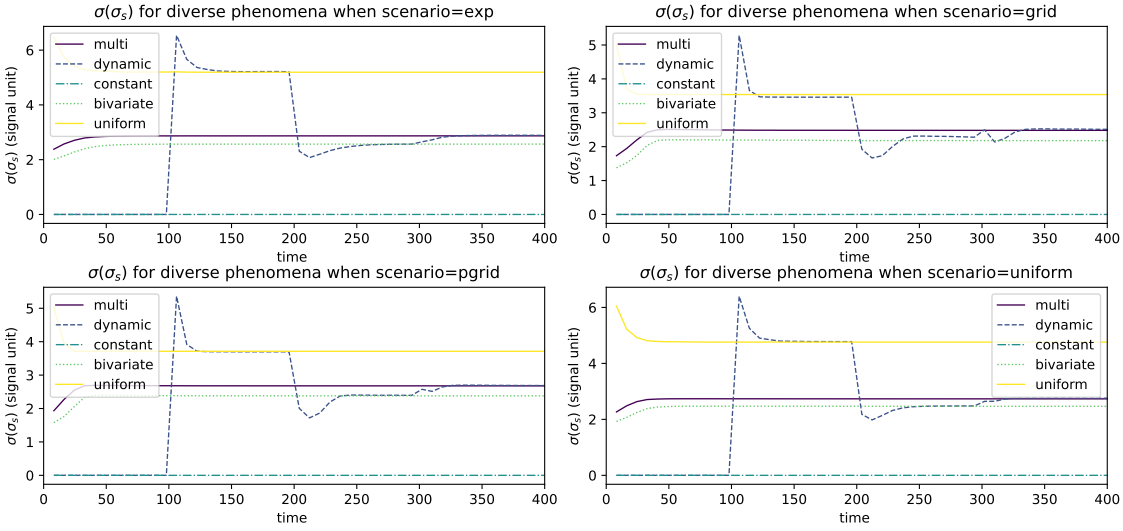


FIGURE 13. Intra-region partitioning homogeneity, measured as the standard deviation across regions of the standard deviation of the signal inside regions. Higher values denote that different regions are more heterogeneous, i.e., that some have larger errors than others.

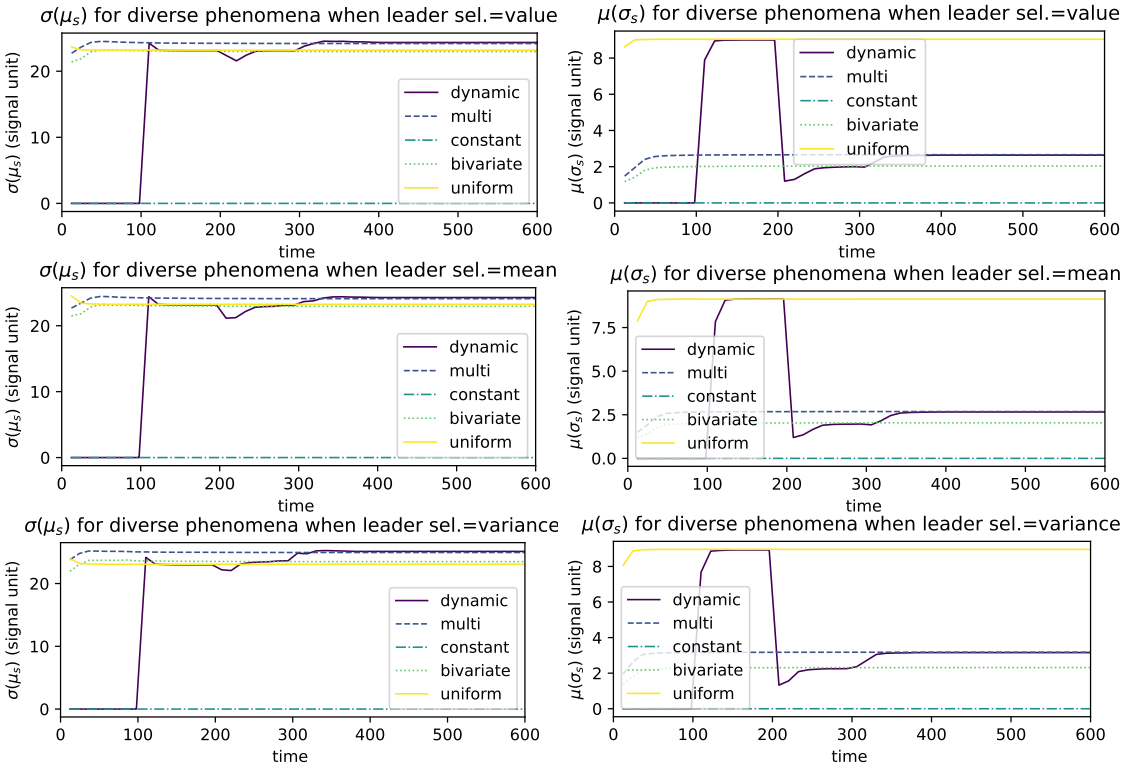


FIGURE 14. Effect of different policies for leader selection. The behaviour of the system is similar regardless of the way the region leader is selected.

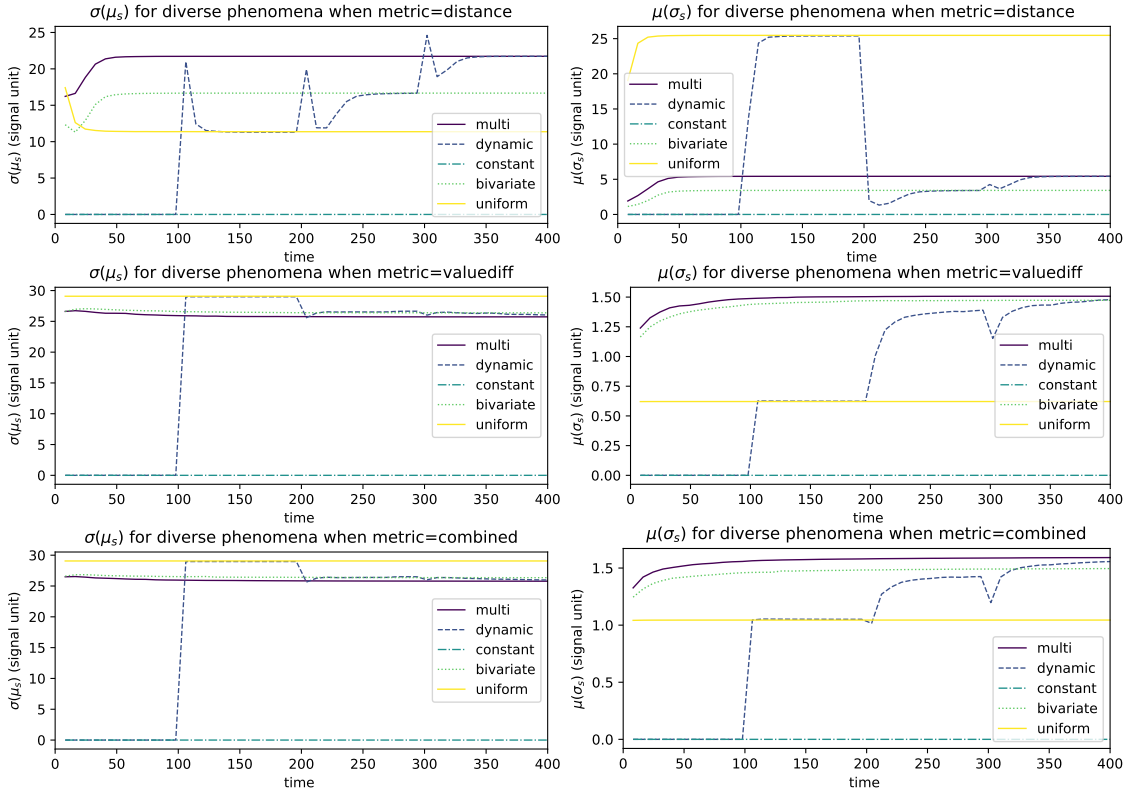


FIGURE 15. Effect of different error measurement metrics. The system is very sensible to the metric used to accumulate error, which directly impacts the way distance is perceived, thus determining the maximum size and number of areas.

5.6. **Results.** The full analysis counts 637 charts; the interested reader can check them out in the experiment repository. In this manuscript, we show the most relevant ones, that we believe help to shed light on the behaviour of the proposed algorithm.

In Figure 11, we show that our implementation stabilises, since after a short transition all values become stable. Obviously, in the dynamic case, these transitions are present throughout the experiment. As expected, the aggregate sampler defines a number of regions that differs depending on the underlying phenomenon under observation. From Figure 12 and Figure 13, we notice that the system indeed tries to maximise inter-region differences and minimise intra-region differences, thus effectively addressing the trade-off between *high information (entropy)* and *error-controlled upscaling* (as per Definition 3.18). Finally, Figure 14 and Figure 15 show how the algorithm reacts to changing parameters. As expected, while modifying the leader selection policy has minimal impact on the behaviour of the system, changing the error-distance metric greatly affects its behaviour. In all cases, we notice that the driver signal with higher information entropy (uniform) generates a larger number of smaller regions than all other signals, while the one with the lowest information entropy (constant) always produces few (usually one) large regions. The reason is that the leader selection impacts the originating point of a region, but it is its expansion (driven by the metric) that ultimately determines its extension and shape.

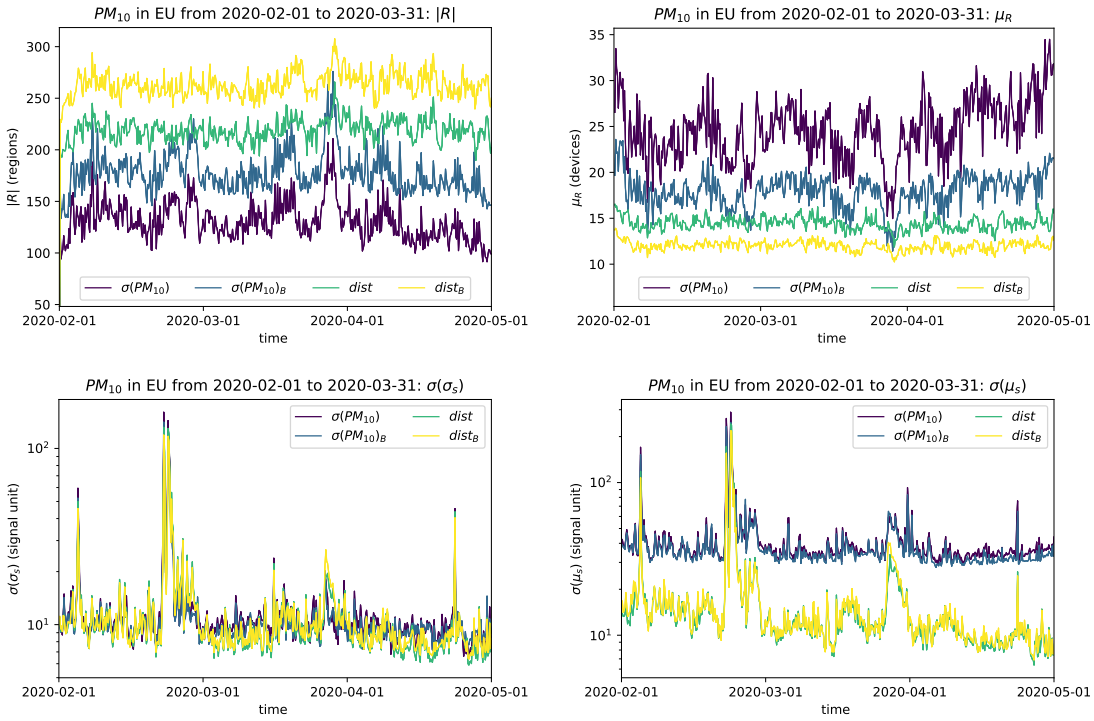


FIGURE 16. Region count (top left), mean region size (top right), intra-region error (bottom left), and inter-region difference (bottom right) with different metrics for the realistic experiment. When fed with a metric that considers both distance and error appropriately, the algorithm is able to create regions in a way that is both efficient and accurate. This is especially true for the ability to maximise the difference among different regions (bottom right). The algorithm is also robust to the presence of arbitrary-set limits, such as the country borders: although introducing them as an impassable barrier generates (with the same metric) a higher region count with a smaller mean size (top charts), the quality of the solution is only marginally affected (bottom charts).

These findings, coming from the analysis of the results produced by the synthetic environments, are confirmed when real-world data is used instead. In Figure 16, indeed, we see that the algorithm is very sensitive to the choice of the metric (as expected); yet, it is pretty robust to the presence of arbitrarily-set spatial limits such as country borders: although they inevitably lead to more regions of a smaller size, the key performance indicators (high inter-regional difference and low intra-regional difference) remain pretty much unchanged.

6. Conclusions and Future Work

In this paper, we tackled the problem of defining an aggregate sampler sensitive to the spatial dynamics of the phenomenon under observation. In particular, we wanted to minimise the

sampling error (minimum when all available sampling devices are used) while also minimising the regions count—two contrasting goals.

We formalised the problem within the framework of event structures and field-based coordination, suitable to represent situated, large-scale, and dynamic computations. We thus designed a spatial adaptive aggregate sampler based on a leader election strategy that dynamically creates and grows/shrinks sampling clusters (or regions) based on two main control "knobs": the error-distance metric and the leader strength. We proved that the proposed algorithm is self-stabilising and enjoys a local optimality property. Through simulation, the proposed algorithm is shown to satisfy the mentioned trade-off.

As measuring performance and efficiency of such an adaptive algorithm is far from trivial, we exploited several metrics to validate intended behaviour. However, as a follow-up work we would like to synthesize a single indicator able to measure both accuracy and efficiency, using information theory such as those derived from entropy (e.g. mutual information). Also, we are analysing openly available air pollution datasets to design new simulations based on real-world data, to better emphasise the impact that our aggregate sampler could have for policy making based on spatial phenomena. Finally, future work will be devoted to investigating how space-fluid sampling can integrate with time-fluid aggregate computations [PCV+21].

Acknowledgment

This work has been supported by the MIUR PRIN 2017 Project "Fluidware" (N. 2017KRC7KT) and the EU FSE PON R&I 2014-2020.

References

[ABD+20] Giorgio Audrito, Jacob Beal, Ferruccio Damiani, Danilo Pianini, and Mirko Viroli. Field-based coordination with the share operator. *Log. Methods Comput. Sci.*, 16(4), 2020. URL: `https://lmcs.episciences.org/6816`.

[ABDV18] Giorgio Audrito, Jacob Beal, Ferruccio Damiani, and Mirko Viroli. Space-time universality of field calculus. In *Coordination Models and Languages - 20th IFIP WG 6.1 International Conference, COORDINATION 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018. Proceedings*, volume 10852 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2018. `doi:10.1007/978-3-319-92408-3_1`.

[ACDV17] Giorgio Audrito, Roberto Casadei, Ferruccio Damiani, and Mirko Viroli. Compositional blocks for optimal self-healing gradients. In *11th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2017, Tucson, AZ, USA, September 18-22, 2017*, pages 91–100. IEEE Computer Society, 2017. URL: `http://doi.ieeecomputersociety.org/10.1109/SASO.2017.18`, `doi:10.1109/SASO.2017.18`.

[AVD+19] Giorgio Audrito, Mirko Viroli, Ferruccio Damiani, Danilo Pianini, and Jacob Beal. A higher-order calculus of computational fields. *ACM Transactions on Computational Logic*, 20(1):1–55, jan 2019. `doi:10.1145/3285956`.

[BC02] Manish Bhardwaj and Anantha P. Chandrakasan. Bounding the lifetime of sensor networks via optimal role assignments. In *Proceedings IEEE INFOCOM 2002, The 21st Annual Joint Conference of the IEEE Computer and Communications Societies, New York, USA, June 23-27, 2002*, pages 1587–1596. IEEE Computer Society, 2002. `doi:10.1109/INFCOM.2002.1019410`.

[BC03] Seema Bandyopadhyay and Edward J. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *Proceedings IEEE INFOCOM 2003, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, San Franciso, CA, USA, March 30 - April 3, 2003*, pages 1713–1723. IEEE Computer Society, 2003. `doi:10.1109/INFCOM.2003.1209194`.

[BC04] Seema Bandyopadhyay and Edward J. Coyle. Minimizing communication costs in hierarchically-clustered networks of wireless sensors. *Comput. Networks*, 44(1):1–16, 2004. doi:10.1016/S1389-1286(03)00320-7.

[BCC+21] Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nowak, Eric E. Severson, and Chuan Xu. Time-optimal self-stabilizing leader election in population protocols. In *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 33–44. ACM, 2021. doi:10.1145/3465084.3467898.

[BPV15] Jacob Beal, Danilo Pianini, and Mirko Viroli. Aggregate programming for the internet of things. *IEEE Computer*, 48(9):22–30, 2015. URL: http://dx.doi.org/10.1109/MC.2015.261, doi:10.1109/MC.2015.261.

[BVPD17] Jacob Beal, Mirko Viroli, Danilo Pianini, and Ferruccio Damiani. Self-adaptation to device distribution in the internet of things. *ACM Transactions on Autonomous and Adaptive Systems*, 12(3):12:1–12:29, 2017. doi:10.1145/3105758.

[CMP+22] Roberto Casadei, Stefano Mariani, Danilo Pianini, Mirko Viroli, and Franco Zambonelli. Space-fluid adaptive sampling: A field-based, self-organising approach. In *Coordination Models and Languages - 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings*, volume 13271 of *Lecture Notes in Computer Science*, pages 99–117. Springer, 2022. doi:10.1007/978-3-031-08143-9_7.

[Cox99] Louis Anthony Cox. Adaptive spatial sampling of contaminated soil. *Risk Analysis*, 19(6):1059–1069, 1999. doi:10.1023/A:1007022409290.

[CPP+20] Roberto Casadei, Danilo Pianini, Andrea Placuzzi, Mirko Viroli, and Danny Weyns. Pulverization in cyber-physical systems: Engineering the self-organizing logic separated from deployment. *Future Internet*, 12(11):203, 2020.

[FRWZ07] Elena Fasolo, Michele Rossi, Jörg Widmer, and Michele Zorzi. In-network aggregation techniques for wireless sensor networks: a survey. *IEEE Wirel. Commun.*, 14(2):70–87, 2007. doi:10.1109/MWC.2007.358967.

[FSM+13] Jose Luis Fernandez-Marquez, Giovanna Di Marzo Serugendo, Sara Montagna, Mirko Viroli, and Josep Lluís Arcos. Description and composition of bio-inspired design patterns: a complete overview. *Nat. Comput.*, 12(1):43–67, 2013. doi:10.1007/s11047-012-9324-y.

[GA14] Sahil Garg and Nora Ayanian. Persistent monitoring of stochastic spatio-temporal phenomena with a small team of robots. In *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*, 2014. URL: http://www.roboticsproceedings.org/rss10/p38.html, doi:10.15607/RSS.2014.X.038.

[GC09] Rishi Graham and Jorge Cortés. Cooperative adaptive sampling via approximate entropy maximization. In *Proceedings of the 48th IEEE Conference on Decision and Control, CDC 2009, combined withe the 28th Chinese Control Conference, December 16-18, 2009, Shanghai, China*, pages 7055–7060. IEEE, 2009. doi:10.1109/CDC.2009.5400511.

[HH17] S. Hoyer and J. Hamman. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1), 2017. doi:10.5334/jors.148.

[HP11] Yousef E. M. Hamouda and Chris I. Phillips. Adaptive sampling for energy-efficient collaborative multi-target tracking in wireless sensor networks. *IET Wirel. Sens. Syst.*, 1(1):15–25, 2011. doi:10.1049/iet-wss.2010.0059.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, 9(3):90–95, May 2007. doi:10.1109/MCSE.2007.55.

[LM07] Yen-Ting Lin and Seapahn Megerian. Sensing driven clustering for monitoring and control applications. In *4th IEEE Consumer Communications and Networking Conference, CCNC 2007, Las Vegas, NV, USA, January 11-13, 2007*, pages 202–206. IEEE, 2007. doi:10.1109/CCNC.2007.47.

[LVP11] Eun Kyung Lee, Hariharasudhan Viswanathan, and Dario Pompili. SILENCE: distributed adaptive sampling for sensor-based autonomic systems. In *Proceedings of the 8th International Conference on Autonomic Computing, ICAC 2011, Karlsruhe, Germany, June 14-18, 2011*, pages 61–70. ACM, 2011. doi:10.1145/1998582.1998594.

[LXZ+13] Zhidan Liu, Wei Xing, Bo Zeng, Yongchao Wang, and Dongming Lu. Distributed spatial correlation-based clustering for approximate data collection in WSNs. In *27th IEEE International*

Conference on Advanced Information Networking and Applications, AINA 2013, Barcelona, Spain, March 25-28, 2013, pages 56–63. IEEE Computer Society, 2013. doi:10.1109/AINA.2013.26.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[MADB20] Yuanqiu Mo, Giorgio Audrito, Soura Dasgupta, and Jacob Beal. A resilient leader election algorithm using aggregate computing blocks. *IFAC-PapersOnLine*, 53(2):3336–3341, 2020. doi:10.1016/j.ifacol.2020.12.1497.

[MCP12] Muhammad F. Mysorewala, Lahouari Cheded, and Dan O. Popa. A distributed multi-robot adaptive sampling scheme for the estimation of the spatial distribution in widespread fields. *EURASIP J. Wirel. Commun. Netw.*, 2012:223, 2012. doi:10.1186/1687-1499-2012-223.

[MHD21] Sandeep Manjanna, Ani Hsieh, and Gregory Dudek. Scalable multi-robot system for non-myopic spatial sampling. *CoRR*, abs/2105.10018, 2021. URL: https://arxiv.org/abs/2105.10018, arXiv:2105.10018.

[MMP+17] Diana Manjarres, Ana Mera, Eugenio Perea, Adelaida Lejarazu, and Sergio Gil-Lopez. An energy-efficient predictive control for HVAC systems applied to tertiary buildings based on regression techniques. *Energy and Buildings*, 152:409–417, October 2017. doi:10.1016/j.enbuild.2017.07.056.

[MR04] Vivek Mhatre and Catherine Rosenberg. Design guidelines for wireless sensor networks: communication, clustering and aggregation. *Ad Hoc Networks*, 2(1):45–63, 2004. doi:10.1016/S1570-8705(03)00047-7.

[MSM18] Hossein K. Mousavi, Qiyu Sun, and Nader Motee. Space-time sampling for network observability. *CoRR*, abs/1811.01303, 2018. URL: http://arxiv.org/abs/1811.01303, arXiv:1811.01303.

[MZ19] Imran Mahmood and Junaid Ahmed Zubairi. Efficient waste transportation and recycling: Enabling technologies for smart cities using the internet of things. *IEEE Electrification Magazine*, 7(3):33–43, 2019. doi:10.1109/MELE.2019.2925761.

[Nag02] Radhika Nagpal. Programmable self-assembly using biologically-inspired multiagent control. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems part 1 - AAMAS'02*. ACM Press, 2002. doi:10.1145/544741.544839.

[NPW81] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theor. Comput. Sci.*, 13:85–108, 1981. doi:10.1016/0304-3975(81)90112-2.

[ÖFL04] Petter Ögren, Edward Fiorelli, and Naomi Ehrich Leonard. Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment. *IEEE Trans. Autom. Control.*, 49(8):1292–1302, 2004. doi:10.1109/TAC.2004.832203.

[PCV+21] Danilo Pianini, Roberto Casadei, Mirko Viroli, Stefano Mariani, and Franco Zambonelli. Time-fluid field-based coordination through programmable distributed schedulers. *Logical Methods in Computer Science*, Volume 17, Issue 4, November 2021. doi:10.46298/lmcs-17(4:13)2021.

[PCV22] Danilo Pianini, Roberto Casadei, and Mirko Viroli. Self-stabilising priority-based multi-leader election and network partitioning. In *IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2022, Virtual, CA, USA, September 19-23, 2022*, pages 81–90. IEEE, 2022. doi:10.1109/ACSOS55765.2022.00026.

[PCVN21] Danilo Pianini, Roberto Casadei, Mirko Viroli, and Antonio Natali. Partitioned integration and coordination via the self-organising coordination regions pattern. *Future Generation Computing Systems*, 114:44–68, 2021. doi:10.1016/j.future.2020.07.032.

[Pia22] Danilo Pianini. Danysk/experiment-2022-coordination-space-fluid: 0.5.0-dev08+67e7add, 2022. URL: https://zenodo.org/record/6473292, doi:10.5281/ZENODO.6473292.

[Pia23a] Danilo Pianini. Aggregated pm10 data for europe, 2023. URL: https://zenodo.org/record/7546591, doi:10.5281/ZENODO.7546591.

[Pia23b] Danilo Pianini. Danysk/experiment-2023-lmcs-pm10-pollution-space-sampling: 2.3.0, 2023. URL: https://zenodo.org/record/7712978, doi:10.5281/ZENODO.7712978.

[PMV13] Danilo Pianini, Sara Montagna, and Mirko Viroli. Chemical-oriented simulation of computational systems with ALCHEMIST. *Journal of Simulation*, 7(3):202–215, 2013. doi:10.1057/jos.2012.27.

[Pra86] Vaughan R. Pratt. Modeling concurrency with partial orders. *Int. J. Parallel Program.*, 15(1):33–71, 1986. doi:10.1007/BF01379149.

[PSS+13] Nathalie Peyrard, Régis Sabbadin, Daniel Spring, Barry W. Brook, and Ralph Mac Nally. Model-based adaptive spatial sampling for occurrence map construction. *Stat. Comput.*, 23(1):29–42, 2013. doi:10.1007/s11222-011-9287-3.

[PVB15] Danilo Pianini, Mirko Viroli, and Jacob Beal. Protelis: practical aggregate programming. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 1846–1853, 2015. doi:10.1145/2695664.2695913.

[RHK+05] Mohammad H. Rahimi, Mark H. Hansen, William J. Kaiser, Gaurav S. Sukhatme, and Deborah Estrin. Adaptive sampling for environmental field estimation using robotic sensors. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, Alberta, Canada, August 2-6, 2005*, pages 3692–3698. IEEE, 2005. doi:10.1109/IROS.2005.1545070.

[SAL+03] John A. Stankovic, Tarek F. Abdelzaher, Chenyang Lu, Lui Sha, and Jennifer C. Hou. Real-time communication and coordination in embedded sensor networks. *Proc. IEEE*, 91(7):1002–1022, 2003. doi:10.1109/JPROC.2003.814620.

[SGAP00] Katayoun Sohrabi, Jay Gao, Vishal Ailawadhi, and Gregory J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Wirel. Commun.*, 7(5):16–27, 2000. doi:10.1109/98.878532.

[SKS10] Piotr Szczytowski, Abdelmajid Khelil, and Neeraj Suri. Asample: Adaptive spatial sampling in wireless sensor networks. In *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, SUTC 2010 and IEEE International Workshop on Ubiquitous and Mobile Computing, UMC 2010, 7-9 June 2010, Newport Beach, California, USA*, pages 35–42. IEEE Computer Society, 2010. doi:10.1109/SUTC.2010.37.

[Tho90] Steven K. Thompson. Adaptive cluster sampling. *Journal of the American Statistical Association*, 85(412):1050–1059, December 1990. doi:10.1080/01621459.1990.10474975.

[VAB+18] Mirko Viroli, Giorgio Audrito, Jacob Beal, Ferruccio Damiani, and Danilo Pianini. Engineering resilient collective adaptive systems by self-stabilisation. *ACM Transactions on Modeling and Computer Simulation*, 28(2):1–28, mar 2018. doi:10.1145/3177774.

[VBD+19] Mirko Viroli, Jacob Beal, Ferruccio Damiani, Giorgio Audrito, Roberto Casadei, and Danilo Pianini. From distributed coordination to field calculus and aggregate computing. *Journal of Logical and Algebraic Methods in Programming*, 109:100486, December 2019. doi:10.1016/j.jlamp.2019.100486.

[VS05] Reino Virrankoski and Andreas Savvides. TASC: topology adaptive spatial clustering for sensor networks. In *IEEE 2nd International Conference on Mobile Adhoc and Sensor Systems, MASS 2005, November 7-10, 2005, The City Center Hotel, Washington, USA*, page 10. IEEE Computer Society, 2005. doi:10.1109/MAHSS.2005.1542850.

[WH07] Tom De Wolf and Tom Holvoet. Designing self-organising emergent systems based on information flows and feedback-loops. In *Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007, Boston, MA, USA, July 9-11, 2007*, pages 295–298. IEEE Computer Society, 2007. doi:10.1109/SASO.2007.16.

[WKT11] Fang-Jing Wu, Yu-Fen Kao, and Yu-Chee Tseng. From wireless sensor networks towards cyber physical systems. *Pervasive Mob. Comput.*, 7(4):397–413, 2011. doi:10.1016/j.pmcj.2011.03.003.

[YF04] Ossama Younis and Sonia Fahmy. Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In *Proceedings IEEE INFOCOM 2004, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, March 7-11, 2004*. IEEE, 2004. doi:10.1109/INFCOM.2004.1354534.

[YVP13] Jing Tao Yao, Athanasios V. Vasilakos, and Witold Pedrycz. Granular computing: Perspectives and challenges. *IEEE Trans. Cybern.*, 43(6):1977–1989, 2013. doi:10.1109/TSMCC.2012.2236648.

[YZMC20] Eun-Hye Yoo, Andrew Zammit-Mangion, and Michael G. Chipeta. Adaptive spatial sampling design for environmental field prediction using low-cost sensing technologies. *Atmospheric Environment*, 221:117091, 2020. doi:10.1016/j.atmosenv.2019.117091.

[ZGMB14] Sabri-E. Zaman, Manik Gupta, Raul J. Mondragón, and Eliane L. Bodanese. An eigendecomposition based adaptive spatial sampling technique for wireless sensor networks. In *IEEE 39th Conference on Local Computer Networks, LCN 2014, Edmonton, AB, Canada, 8-11 September, 2014*, pages 430–433. IEEE Computer Society, 2014. doi:10.1109/LCN.2014.6925809.