IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

A field-based computing approach to sensing-driven clustering in robot swarms

This is a pre print version of the following article:

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/1889559 since 2023-06-02T11:50:49Z

Published version:

DOI:10.1007/s11721-022-00215-y

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

12 October 2024

A Field-based Computing Approach to Sensing-driven Clustering in Robot Swarms

Gianluca Aguzzi[1], Giorgio Audrito[2], Roberto Casadei[1*], Ferruccio Damiani[2], Gianluca Torta[2] and Mirko Viroli[1]

[1*]Department of Computer Science and Engineering, Alma Mater Studiorum – Università di Bologna, Via dell'Università, 50, Cesena, 47521, Italy.
[2]Department of Computer Science, Università degli Studi di Torino, C.so Svizzera, 185, Torino, 10149, Italy.

*Corresponding author(s). E-mail(s): roby.casadei@unibo.it; Contributing authors: gianluca.aguzzi@unibo.it; giorgio.audrito@unito.it; ferruccio.damiani@unito.it; gianluca.torta@unito.it; mirko.viroli@unibo.it;

Abstract

Swarm intelligence leverages collective behaviours emerging from interaction and activity of several "simple" agents to solve problems in various environments. One problem of interest in large swarms featuring a variety of sub-goals is *swarm clustering*, where the individuals of a swarm are assigned or choose to belong to zero or more groups, also called *clusters*. In this work, we address the *sensing-based* swarm clustering problem, where clusters are defined based on both the values sensed from the environment and the spatial distribution of the values and the agents. Moreover, we address it in a setting characterised by decentralisation of computation and interaction, and dynamicity of values and mobility of agents. For the solution, we propose to use the field-based computing paradigm, where computation and interaction are expressed in terms of a functional manipulation of *fields*, distributed and evolving data structures mapping each individual of the system to values over time. We devise a solution to sensing-based swarm clustering leveraging multiple concurrent field

computations with limited domain, and evaluate the approach experimentally by means of simulations, showing that the programmed swarms form clusters the underlying environmental phenomena dynamics.

Keywords: sensing-based clustering, swarm clustering, computational fields, multi-agent cluster formation

1 Introduction

Swarm intelligence is the collective-level ability to solve problems in large groups of relatively simple agents that interact with each other locally, i.e., based on physical/logical proximity (Bonabeau et al, 1999). Swarm intelligence is a phenomenon observed both in natural systems (cf. social insects and animals) and artificial systems (cf. computational ecosystems) (Bonabeau et al, 1999). In computer science and engineering, research fields like *swarm robotics* (Brambilla et al, 2013) and self-organising systems (Serugendo et al, 2011, 2007) emerged to study algorithms, models, and techniques for promoting swarm intelligence in artificial systems for a variety of contexts and applications including (but not limited to) environment monitoring (De Masi and Ferrante, 2020; Casadei et al, 2020a), enterprise software service coordination (Clark et al, 2015), crowd management (Beal et al, 2015), and most specifically control of robot swarms (groups of relatively simple robots) (Shen et al, 2004; Carrillo-Zapata et al, 2018). A common distinction is between *behaviour-based* and *automatic* design methods (Brambilla et al, 2013): the former is based on a manual specification of individual behaviour, whereas in the latter the individual behaviour is generated automatically, by searching, adapting, or evolving individual behaviours for effective collective behaviour. Common but not exhaustive classes of collective behaviours include spatial organisation (e.g., pattern formation), swarm navigation, and collective-decision making (Brambilla et al, 2013).

In particular, one problem of interest is *swarm clustering* (Lee et al, 2005; Cruz et al, 2017), whereby the classical data clustering task (i.e., the unsupervised learning task where data items are grouped to promote intra-group similarity) is brought in swarm settings. This problem revolves around splitting the swarm into groups of individuals, called *clusters*, such that the individuals in the same *cluster* are more similar to each other (for some definition of *similarity*) than to those in other clusters. Once a cluster is formed, typically it is assigned a sub-goal to be carried on collectively. Typical clustering approaches may consider the spatial distribution of the individuals or the goals of the individuals to define clusters teams or interaction domains. In this paper, we focus on *sensing-based clustering* (Lin and Megerian, 2007), namely a clustering problem that considers both the spatial distribution of individuals and the environmental values sensed by these individuals (through sensors). That is, the goal is to seek for clusters of neighbour individuals with a similar perception of some sensed value. The problem can be in a *static* form, where a snapshot of the system state is considered, or in a *dynamic* form, where values change over time and solutions have to deal with change somehow. The problem has been considered in Wireless Sensor Networks (WSNs) and Internet-of-Things (IoT) applications like environment monitoring and control (Lin and Megerian, 2007), efficient distributed collection (Pham et al, 2010), and disaster management (Kucuk et al, 2020). However, to the best of our knowledge no existing work addresses the dynamic problem in mobile swarms, which requires

specific techniques to adaptively re-adjust clusters to face changes. Accordingly, in this work, we present and address the *dynamic sensing-based swarm clustering* problem.

Among the many approaches to express (and reason in terms of) collective behaviour featuring inherent adaptivity we shall consider the *field-based computing* approach (Viroli et al, 2019), for its suitability in addressing dynamic problems by fostering "controlled self-organisation". In this approach, computations leverage an execution model based on repeated computation and asynchronous neighbour-based communication. On top, complex collective behaviour is described in terms of functional manipulations of *(computational) fields*, i.e., data structures evolving over time that map agents in a domain to computational values—sort of spatially distributed streams of values. This is inspired by the common notion of fields found in physics (e.g., force or magnetic fields). Notice, however, that in our viewpoint, the computational fields assign values to agents rather than to environment (space-time) positions as in e.g. *artificial potential fields* (Warren, 1989), though the approaches are similar and related. We adopt this approach as it has shown to conveniently express a variety of collective swarm-like behaviour including self-healing distance estimation (*gradient*) (Audrito et al, 2017), self-stabilising leader election (Mo et al, 2018), distributed collection (Audrito et al, 2021), and team creation and coordination (Casadei et al, 2021)—and to scale with complexity up to high-level composite patterns (Pianini et al, 2021b).

Essentially, the core idea of our clustering approach is to make agents in local of the sensed value spawn a spatial process of gathering for neighbour devices until finding the proper size of the cluster, additionally managing interactions with other clusters when there are overlaps.

In this manuscript we provide the following contributions:

- we provide a precise definition of the dynamic sensing-based mobile swarm clustering problem;
- we present a field-based approach to address the problem, and describe a novel configurable meta-algorithm for inducing self-organised clustering in a system of neighbouring-interacting robots;
- we provide a publicly available and reproducible simulation framework for evaluating the algorithm on a set of diverse environment configurations, from which we observe that our solution can identify various cluster shapes and cope with a certain degree of node mobility and changes in sensed phenomena.

Therefore, the contribution lies both in the general of swarm intelligence as well as in the specific thread of research in field-based computing.

The paper is organised as follows. Section 2 covers background, introducing the field-based computing paradigm and the swarm clustering problem. Section 3 provides the novel technical contribution. Section 4 presents our evaluation of the approach. Section 5 covers related work. Finally, Section 6 provides a summary and discusses future research directions.

2 Background and Motivation

The background of this work includes field-based computing (Section 2.1) and the problem of clustering in swarms (Section 2.2).

2.1 Field-based Computing

Field-based computing (Viroli et al, 2019) is an approach where computation leverages a notion of *computational fields* (*fields* for short) (Warren, 1989; Mamei et al, 2004; Viroli et al, 2019), namely distributed data structures evolving in time and associating locations with values. The approach originates from previous work like Warren's *artificial potential fields* (Warren, 1989) and *co-fields* from Mamei et al. (Mamei et al, 2004). In particular, in co-fields, computational fields represent contextual information, locally sensed by the agents and repeatedly distributed by the agents themselves or the infrastructure according to a propagation rule.

In this work, by field-based computing we mean a specific programming and computational model, also known as *aggregate computing* in literature (Beal et al, 2015), which is surveyed in (Viroli et al, 2019). In this model, collective and self-organising behaviour is programmed through a composition of functions operating on fields mapping a set of individual agents (rather than environment locations) to computational values. Therefore, fields can be used to associate a certain domain of agents with what they sense, the information they process, and actuation instructions for operating on the environment. Fields are computed locally to the agents but are subject to a global viewpoint: so, e.g., a field of velocity vectors can be seen as a movement command for an entire swarm, or a field of reals can denote what an entire swarm perceives in a certain environment. To understand field-based computing, two essential parts have to be considered: the system model and the programming model. Their interplay is what allows the local actions of the agents to yield emergent collective behaviour.

2.1.1 System Model

We consider a network of computing and interacting *agents* situated in some *environment*.

Structure. An *agent* is an autonomous entity equipped with *sensors* and *actuators*, which serve as the interface towards a logical or physical *environment*. By a logical point of view[1], it also has *state*, a support for *communicating* with other agents, and support for *computing* simple programs. An agent is connected with other *neighbour* agents which collectively form its *neighbourhood*. The set of neighbours depends on a *neighbouring relationship*, which is defined by designers according to the application at hand and is subject to the constraints exerted by the underlying physical network. A typical neighbouring

[1] Actually, such requirements may be relaxed by considering different execution strategies on available infrastructure (Casadei et al, 2020a).

rule is the one that mimics physical connectivity; so, e.g., a robot is a neighbour of another robot if it manages to send a message to the latter over the wireless channel. Another typical neighbouring rule is the one based on spatial vicinity; so, e.g., a robot is a neighbour of another robot if the infrastructure manages to deliver a message from the former to the latter (e.g., using other robots as relays) and these two robots are at an estimated distance smaller than a certain threshold (assuming a distance can be estimated through a proper technology).

Interaction Interaction happens by sending messages to neighbours, asynchronously. Interaction can also happen in a stigmergic way, by perceiving and acting upon the environment through sensors and actuators. The content of messages and when they are sent and received depend on the agent behaviour. However, in general, as our goal is to model continuous collective behaviours, or self-organising systems, we remark that interaction would typically be frequent (in relation to the problem and environment dynamics).

Behaviour. As per the above consideration, the behaviour of any individual agent is best understood in terms of repeated enaction of *execution rounds*, where each round consists of the following steps (though some flexibility exists especially in the actuation part):

1. *Context acquisition.* The agent gathers its context by considering its previous state as well as the most recent sensor readings and messages from neighbours.
2. *Computation.* The agent runs a computation against the acquired context, yielding (i) an *output* describing potential actuations; and (ii) a *coordination message* containing all the information to be sent to neighbours for the purpose of coordination at a collective level.
3. *Actuation .* The agent performs the actuations described by the program output and dispatches the coordination message to the entire neighbourhood.

This system model provides a basic machinery for collective adaptive behaviour, which however requires a proper description of the "local computation step": this is fostered by the *field-based programming model* (discussed in Section 2.1.2).

2.1.2 Field-based Programming Model

Field-based programs can be encoded with field-based programming languages like ScaFi (Casadei et al, 2020b), . ScaFi is a domain-specific language (DSL) embedded in Scala which supports field-based constructs and offers a library of reusable functions.

A field-based expression or program (e.g., programmed in ScaFi) can be subject to a local or global interpretation. Locally, an integer value like `7` has the usual meaning; globally, a `7` denotes a field where each agent is mapped to a local `7` (a uniform, constant field).

Locally, an integer expression `add(a,b)`, or `a+b`, has the usual meaning, given by the sum of `a` with `b`; globally, it denotes the application of a field of functions `add`, or `+`, on a field `a` and a field `b`, yielding a field given by the sum of `a` and `b` in an agent-wise fashion (notice that `a` may be a non-uniform non-constant field having different local values for different agents over time).

The programming model does not deal directly with global fields (which are essentially a denotational construct), but it deals only with *neighbouring fields*, which enable one agent to collect data from its neighbours.

in the following, we briefly present a subset of the field-based computing building blocks used for sensing-based clustering. See (Viroli et al, 2019) for more details on how these blocks are actually developed.

Typically, in field-based computing applications, we are dealing with sharing and collecting information from/to a device.

To do this, the `gradient` is an essential construct (Audrito et al, 2017). This block produces a numeric field that expresses the minimum distance from a source zone following a certain metric (e.g., Euclidean distance). Hence, it maps a Boolean field (`true` where a node is a source, `false` otherwise) into a distance field from the closest source. The signature of the function is defined as[2]:

```
def gradient(source: Boolean, metric: Metric): Double
```

hrough this construct, it is possible to share generic data (a position, a temperature, etc.) towards this resulting distance field. Such propagation of data from a source of a gradient outwards is captured by a `broadcast` function :

```
def broadcast[D](source: Boolean, data: D): D
```

When we want to aggregate data in source agents, we use the block `C` (collect) instead (Audrito et al, 2021):

```
def C[V](p: Double, acc: (V, V) => V, local: V, null: V): V
```

In this signature, `p` is a potential field usually computed through `gradient`; `acc` is the logic that combines ; `local` is the local data we want to collect at a point in space (e.g. a position); and `null` is the null data for the `acc` operation (e.g. if we collect a real value, the `null` value could be 0). This is also an essential operation for the definition of collective behaviours: it enables, e.g. computation of the average temperature in a certain zone covered by agents.

```
val leader: Boolean = // true on leader devices
val potential: Double = gradient(leader, metric())
val collect: Int = C[Int](potential, (sum,v)=>sum+v, 1, 0)
val count: Int = broadcast[Int](leader, collect)
```

2

2.1.3 Field-based Concurrent Processes

Field-based concurrent processes, also called *aggregate processes* (Casadei et al, 2019, 2021), are field-based computations that exist dynamically: they can be dynamically generated (usually by individual agents), execute on a dynamic set of agents, and disappear once all its members withdraw. They have been formalised in (Casadei et al, 2019) and deeply covered in (Casadei et al, 2021), showing how they can support the design of intelligent collective behaviour by extending the practical expressiveness of field-based programming models (Viroli et al, 2019). We provide a brief account of the details relevant for this manuscript in the following.

Indeed, the aggregate process abstraction is relevant in this work since an aggregate process instance, by running on a (evolving) subset of the agents, can be used to denote a dynamic cluster. Therefore, clustering algorithms can be expressed in terms of how aggregate processes are generated (candidate cluster formation) and merged/removed (cluster selection).

Aggregate processes can be expressed as normal field-based functions and spawned through a `spawn` construct with the following signature:

```
// spawn is a generic function which accepts 3 parameters
def spawn[K,A,R](process: K => A => (R,Boolean),
                 newProcesses: Set[K],
                 args: A): Map[K,R]
```

The generic type `K` instantiates to the type of a *process key*, also called a *process identifier (PID)*, which also works as construction parameter; the generic type `A` instantiates to the type of runtime parameters for the currently running process instances; the generic type `R` instantiates to the type of the output of the process. A *process definition* has curried type `K => A => (R,Boolean)`, namely a function from a value of type `K` and a value of type `A` to a pair of a value of type `R` and a Boolean. The Boolean value, called the *process status*, expresses if the device that has executed a given process instance would like to participate into the process (`true` status) or not (`false` status). The crucial point is that every device that participates into a process with PID π automatically propagates the process PID π to all its neighbours, which will run a corresponding process instance when the `spawn` function is evaluated. So, the `spawn` function accepts a function `process` of a field-based behaviour, a set `newProcesses` of new process instances to be generated locally in the current round, and a value of type `A` for the runtime input of the instances currently running in the local round of a given device. Notice that, though `process` can be a field of functions, it is typically a constant field of the same function, which means that usually a `spawn` expression enables running zero or more process instances of the same kind of process. Evaluation of `spawn` returns a `Map[K,R]` (i.e., a hashmap or dictionary) which a set of entries mapping the PIDs of executed process instances (with status `true`) to corresponding outputs of type `R`.

As an example, consider building a separate gradient computation for each distinct source agent, that will expand within a certain range ρ. This could be coded as follows in ScaFi:

```
type DeviceId = Int
// Process definition as a function
val proc: DeviceId => Boolean => (Double, Boolean) = id => isSource => {
  val output = gradient(id == deviceId())
  val status = if(id == deviceId()) isSource
               else output < ρ
  (output, status)
}
// Set of processes to be generated locally
val newProcesses: Set[DeviceId] =
  if(isSource()) Set(deviceId()) else Set.empty
// Expression for handling acquired and generated processes
val gradients: Map[DeviceId,Double] =
  spawn[DeviceId,Boolean,Double](process, newProcesses, isSource())
```

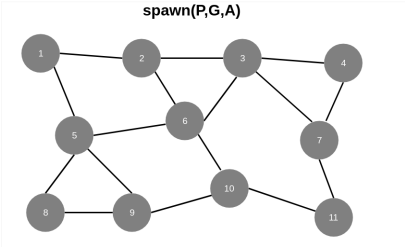
An example of the dynamics of such a program is provided in Figure 1. In the picture: nodes are agents; labels on nodes are agent IDs; edges denote neighbouring links, over which messages are sent and received; the output of the `spawn` expression is shown above the nodes, unless it is an empty map (not shown); the different sub-pictures are snapshots of a corresponding hypothetical system state trajectory that may result after multiple rounds of execution in multiple devices. A more thorough introduction and description of aggregate processes together with more examples is available in (Casadei et al, 2021).

2.2 Dynamic Cluster Formation in Swarms

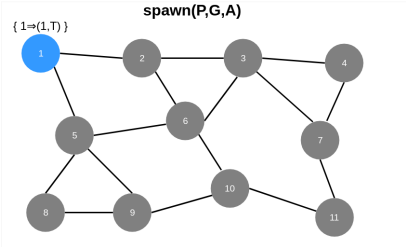
Different cluster models exist and, for each cluster model, several algorithms can be devised (Estivill-Castro, 2002). These are reviewed and compared with our cluster model in Section 5.

In this paper, we focus on *swarm clustering*, which involves associating each swarm member to zero or more clusters. So, this is a problem of *cluster formation* (Ge et al, 2018), more than a problem of *cluster analysis* (which generally includes cluster formation followed by cluster evaluation). A *cluster*, in this setting, is essentially a label (*cluster ID*), which can be associated to an agent, and that can be used to determine its behaviour. In field terms, a clustering can be seen as a field mapping each agent to a set of cluster IDs—we call this a *clustering field*.

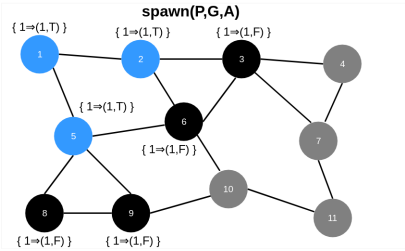
Essentially, a cluster can be used to determine, query, and control a group of agents. Such a group could represent a team, used for cooperation or to solve a common goal, or a space-time domain for a field computation. Indeed, as the agents are situated in space, they provide a means for extracting data from their corresponding location, which may be instrumental for environment monitoring, data acquisition, etc.



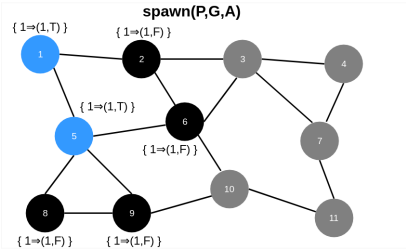
(a) Initial network.



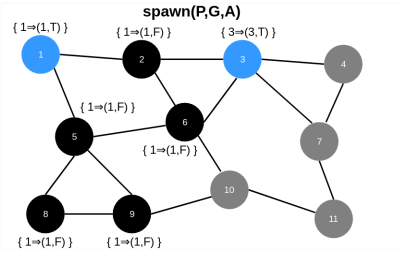
(b) A process is generated on agent 1.



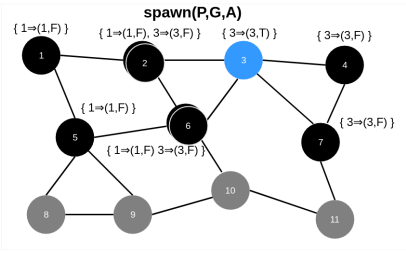
(c) The process with PID 1 propagates up to a certain range.



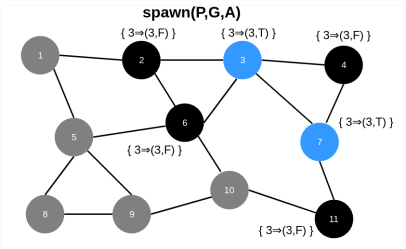
(d) The "border" of a process can change dynamically.



(e) Another process is spawned by source agent 3.



(f) Processes can overlap. Agents 2 and 6 run the two processes with PID 1 and 3.



(g) Process 1 ceases to exist.

Fig. 1: Examples of the dynamics of multiple concurrent gradient processes.

Moreover, we consider *dynamic clustering* (Roa et al, 2019), where the emphasis is not on identifying a single clustering for a given system configuration, but to update and evolve a clustering solution as the system configuration evolves (e.g., due to mobility, failure, or change in other clustering criteria). The specific problem we tackle is *dynamic sensing-based/space-based swarm clustering*, which involves associating each swarm member to zero or more clusters, and to evolve such association by considering change in the environment (*sensing-based*) and spatial location of the members (*space-based*).

In summary, our goal is to define a distributed, decentralised, field-based clustering algorithm, for the system model described in Section 2.1.1, able to create and dynamically maintain a clustering field. This work draws motivation from (i) the relevance of the problem for situated systems (e.g., in swarm robotics), (ii) a scarcity of solutions to the problem of *sensing-driven spatial clustering* in literature, and (iii) a general lack of effective field-based clustering solutions. Refer to Section 5 for a more detailed account on these research gaps.

3 Contribution

3.1 Problem Definition

In this paper, we address the problem of situation awareness and recognition, where a value distributed in space (e.g., temperature as measured by sensors) has to be monitored, by recognising compact clusters with similar values (e.g., spatial regions with a similar temperature). This problem, called *sensing-driven clustering* in literature, has been investigated largely in static scenarios (Kucuk et al, 2020; Pham et al, 2010; Lin and Megerian, 2007), where data from a fixed sensor network has to be processed in order to obtain the relevant clusters. However, solutions for such networks do not extend well to dynamic contexts, such as micro-drone swarms monitoring an environment: in this scenario, mobility and proximity of communication are key, and need to be handled by an algorithm that is resilient to changes in both values, network structure and placement in space. To the best of our knowledge, this problem has never been previously considered in the literature.

A sensing-driven clustering algorithm for mobile could be useful for several outcomes. Clusters may provide a compressed summary of the value distribution in space, to upload on the cloud and be graphically represented for human convenience. Clusters may also be used to drive more complex situation recognition patterns: algorithms to detect dangerous situations may be run in each cluster separately, using information from that cluster to reach a verdict, without interference from information on neighbouring clusters. Clusters may also be used to drive task assignment to the monitoring drones, possibly guiding their placement in space, by directing more drones in clusters where the need arises.

More formally, we consider the following problem:

- **Input:** for each device, a unique identifier i and a *value* v_i of type T

- **Output:** for each device, a list of clusters to which the device belongs, represented as a map from unique identifiers l of cluster leaders to corresponding cluster summary values w^l of type S.

In order to formally specify the output, we need some further details characterising what a *cluster* is, how they should be selected, and what is their *summary*. This is attained through the following problem parameters.

- **Metric:** a data type M with
  - a *null* value 0_M;
  - a partial order[3] $x \leq y$ defined for x, y of type M;
  - an addition operator $x + y$ defined for x, y of type M, such that $x + 0_M = x$ and $x + y > x$ if $y > 0_M$;
  - a positive function $d(i, j) > 0_M$ returning a value in M representing a distance between a device i and j (depending on the devices' sensor states and possibly values v_i).
- **Summary:** a data type S with
  - a value $s(i)$ of type S in every device i (depending on sensor state);
  - an associative and commutative function $f : (S, S) \to S$, used to aggregate values $s(i)$ for devices in a same cluster.
- **Leader selection:**
  - a candidate radius $r(i)$ in M (depending on sensor state and values), so that only devices with a relative distance strictly lower than $r(i)$ can belong to a cluster whose leader is i;[4]
  - a commutative similarity predicate $p : (S, S) \to \{\top, \bot\}$, identifying similar clusters based on their summary.

According to this description, a candidate cluster $\mathcal{C}$ is a set of devices with a leader i, such that every $j \in \mathcal{C}$ is within a distance of $r(i)$ from the leader i, according to the metric given by d. The summary w_i of such cluster is the repeated aggregation through f of the values $\{v_j : j \in \mathcal{C}\}$. Nearby clusters are merged if their summaries are similar according to predicate p, and in such case, the lowest identifier is selected as the leader of the merged cluster.

We highlight that we aim to solve this problem by an *adaptive* algorithm, that is, a program that is able to handle changes in its input, by periodically and asynchronously updating its internal values.

3.2 Adaptive Centroid-based Clustering on Numeric Values

In the evaluation section, we consider a specific instantiation of the for centroid-based clustering on numeric values. In this context, the metric is a simple distance on values, so that $d(i, j) = |v_i - v_j|$. To prevent the creation of a candidate cluster for every device, the candidate radius $r(i)$ is set to zero whenever i is not a local minimum (i.e., has a neighbouring device j such that $v_j < v_i$). If instead i is a local minimum, $r(i)$ is set to a fixed difference

[3] A partial order is a reflexive, transitive and anti-symmetric relation; with no requirement that either $x \leq y$ or $y \leq x$ for x, y of type M.

[4] Notice that $r(i) = 0_M$ implies that no device can be in a cluster whose leader is i.

i	current device	$\mathcal{N}(i)$	neighbour set
ℓ	candidate leader	$\mathcal{S}_i$	candidate leader set
m_i^ℓ	metric in i from ℓ	c_i^ℓ	whether i belongs to cluster ℓ
p_i^ℓ	parent of i in cluster ℓ	t_i^ℓ	partial summary in i for cluster ℓ
u_i^ℓ	candidate leader summary in i for ℓ		
l_i	selected leader for cluster i, if any	w_i	selected summary for cluster i, if any
l_i^ℓ	selected leader for cluster ℓ in i	w_i^ℓ	selected summary for cluster ℓ in i

Table 1: State variables used in the state equations.

value θ. The values $s(i)$ to be summarised are set to a tuple $[x_i, y_i, v_i, 1]$ of the devices' positions[5] and values with the number 1, with an aggregator function f that is a component-wise sum, so that the overall aggregate of a cluster $\mathcal{C}$ is (eventually) equal to the tuple $[\sum_{i\in\mathcal{C}} x_i, \sum_{i\in\mathcal{C}} y_i, \sum_{i\in\mathcal{C}} v_i, \#\mathcal{C}]$ (where $\#\mathcal{C}$ is the actual number of members of cluster C). The similarity predicate p then declares two clusters as similar if they have centroids within a radius of γ, in a 3D space mixing spatial coordinates with a value coordinate:

$$p([x, y, v, n], [x', y', v', n']) \quad := \quad \|\frac{(x, y, v)}{n} - \frac{(x', y', v')}{n'}\| < \gamma$$

where (x, y, v) denotes a 3D vector and $\|\cdot\|$ denotes the norm of a vector. By setting the problem parameters as described, the meta-algorithm can select clusters of similar value, led by their minima, and merge overlapping clusters that are too close together and with a similar value.

3.3 Adaptive Clustering Meta-Algorithm

We now describe meta-algorithm for the stated problem through state equations. The algorithm state is distributed, hence composed of variables x_i depending on a device identifier i: we assume that such a variable is stored in device i and periodically updated by it through the state equations. Each equation may involve inspecting the state of variables in neighbour devices j: we assume that every device periodically shares its state with neighbours, so that a (not necessarily updated) view of neighbours' state is available in each device, and each state equation can be computed locally in the current device i, without remote memory accesses. We use $\mathcal{N}(i)$ to denote the set of current neighbours of device i, i.e., the set of devices j for which a view of their state is locally available in i (not including i itself). The execution of state equations can be performed in asynchronous rounds, as described in Section 2.1.

Table 1 summarises the state variables used in state equations. Every device maintains a candidate leader set $\mathcal{S}_i$, of possible clusters to which the device may belong. Every round, this set is updated as:

$$\mathcal{S}_i \;=\; \{\ell \in \mathcal{S}_j \text{ for } j \in \mathcal{N}(i) \text{ s.t. } c_j^\ell = \top\} \cup \begin{cases} \emptyset & \text{if } r(i) = 0_M \\ \{i\} & \text{otherwise} \end{cases}$$

[5] We assume that a GPS-like sensor is available.

Thus, $\mathcal{S}_i$ includes i provided that $r(i) > 0_M$, together with other candidate leaders ℓ considered by neighbours (in their candidate leader set and which have computed to be within the cluster). In field-based computing, this set is implicitly maintained by the *spawn* construct, given c_i^ℓ as process return status and $\{i\}$ as new process key (if $r(i) > 0_M$).

Most of the meta-algorithm computation is repeated for each of the candidate leaders $\ell \in \mathcal{S}_i$. First, a metric m_i^ℓ of distance between ℓ and i is computed, through the following equation (called the `gradient` block in field-based computing—cf. Section 2.1):

$$m_i^\ell = \begin{cases} 0_M & \text{if } \ell = i \\ \min\{m_j^\ell + d(i,j) : \ j \in \mathcal{N}(i)\} & \text{otherwise} \end{cases}$$

Then, an optional parent p_i^ℓ for $\ell \neq i$ is determined as the neighbour j with minimal m_j^ℓ (resolving ties by the identifier j itself):

$$p_i^\ell = \begin{cases} \arg\min_{j \in \mathcal{N}(i)}\{(m_j^\ell, j)\} & \text{if } \ell \neq i \\ \text{None} & \text{otherwise} \end{cases}$$

Through it, partial summaries t_i^ℓ can be computed (`C` block in field-based computing—cf. Section 2.1):

$$t_i^\ell = \text{reduce}(\{s(i)\} \cup \{t_j^\ell : \ j \in \mathcal{N}(i) \text{ and } p_j^\ell = i\}, f)$$

where "reduce" is a function accumulating every element of a given set with the given binary function, and thus aggregates with f the value $s(i)$ together with the t_j^ℓ values of neighbours j which chose the current device i as their parent. The value of the partial summary in the leader is then propagated through the cluster by a broadcast function:

$$u_i^\ell = \begin{cases} t_i^\ell & \text{if } \ell = i \\ u_{p_i^\ell}^\ell & \text{otherwise} \end{cases}$$

Every candidate leader i with $r(i) > 0_M$ is now able to choose its selected leader l_i, as the minimum candidate leader j (possibly i itself) with a summary similar to that of i according to predicate p:

$$(l_i, w_i) = \begin{cases} \min\{(\ell, u_i^\ell) : \ \ell \in \mathcal{S}_i \text{ and } p(u_i^\ell, u_i^i)\} & \text{if } r(i) > 0_M \\ \text{None} & \text{otherwise} \end{cases}$$

```
// process starts when r(i) is positive
val newProc = mux (r(i) > 0) { Set(mid) } { Set.empty }
// collect map from ℓ ∈ to (m_i^ℓ, u_i^ℓ)
val clusters = spawn(ℓ => _ => {
  val m_i^ℓ = gradient(mid == ℓ, d) // distance estimation
  val c_i^ℓ = m_i^ℓ < r(ℓ) // whether device is in cluster
  val t_i^ℓ = C(m_i^ℓ, f, s(i)) // summary collection
  val u_i^ℓ = broadcast(m_i^ℓ, t_i^ℓ) // summary broadcast
  return ((m_i^ℓ, u_i^ℓ), c_i^ℓ) // process result and status
}, newProc, ())
// selected leader
val l_i = mux (r(i) > 0) {
  clusters.filter(x => p(x._2, clusters(mid))).keys.min
} { mid }
// selected leader summary
val w_i = mux (r(i) > 0) { clusters(l_i)._2 } { None }
// propagate in process
val result = spawn(ℓ => _ => {
  val m_i^ℓ = clusters(ℓ)._1 // recover distances
  val c_i^ℓ = m_i^ℓ < r(ℓ) // whether device is in cluster
  val (l_i^ℓ, w_i^ℓ) = broadcast(m_i^ℓ, (l_i, w_i)) // final broadcast
  return ((l_i^ℓ, w_i^ℓ), c_i^ℓ) // process result and status
}, newProc, ())
// build result map
return result.map(x => { x._2._1 -> x._2._2 })
```

Fig. 2: Scafi pseudo-code of the meta-algorithm.

The selected leader l_i and corresponding summary w_i is then propagated by broadcast through the cluster of i. For every $\ell \in \mathcal{S}_i$:

$$(l_i^\ell, w_i^\ell) = \begin{cases} (l_i, w_i) & \text{if } \ell = i \\ (l_{p_i^\ell}^\ell, w_{p_i^\ell}^\ell) & \text{otherwise} \end{cases}$$

Finally, in every device i, the meta-algorithm output is the map:

$$\{l_i^\ell \mapsto w_i^\ell : \ \ell \in \mathcal{S}_i\}.$$

This meta-algorithm is presented as ScaFi pseudo-code in Figure 2, using ScaFi library functions `gradient`, `C`, and `broadcast`—cf. Section 2.1. We also remark that although values v_i are not directly used by the meta-algorithms, the parameters $r(i)$ and $d(i,j)$ are allowed to depend on them (and usually do), so that values are indirectly used. An example of such behaviour is given in the next section.

4 Evaluation

In this section, we evaluate the meta-algorithm proposed in Section 3.3 in a case study of situation recognition within a synthetic environment . The goal is to show how the algorithm can cluster agents in a sensing-based fashion, hence identifying various temperature cluster shapes. Furthermore, we assess how the algorithm works in mobile settings, where a swarm of agents moves across an environment—which can be representative for exploration scenarios.

4.1 Scenario Description

A swarm group of robots is interested in identifying areas where environmental data varies within a known range. In particular, we assume that the robots are both capable of sensing the environmental temperature, perceiving their position in space (e.g., using GPS), and exploring a limited area (i.e., a square with a side of 1 km). The temperature is just an arbitrary choice of a sensible physical quantity that should drive, together with the spatial distribution, the clustering; the idea is that a temperature can be indicative for an environment situation that could require attention or intervention (cf. wildfires which can start and spread in hot, dry, and windy conditions). The scenarios are plausible, but we are not interested in full realism: simplifications and generalisations are introduced to study the algorithm in diverse controlled situations. Since the absence of central authority and the limited robot communication capability, we suppose that the robots can only interact with their neighbours (i.e., the devices with which a robot manages to establish a connection). In particular, we imagine that each robot is equipped with a LoRa module with a connection range of 100 m. Notice that these assumptions are coherent with the system model of Section 2.1.1.

In the experiments described in the following, we are only interested in the clusters determined by the swarm cooperatively, not in *how* clusters are leveraged at the application level.

4.2 Evaluation Goals

We set up these simulations to:

G.1 verify the capability of the algorithm to find different cluster shapes: we want to check that our algorithm is robust enough to correctly identify any kind of distribution, whether Gaussian or not;

G.2 examine how can cope with drone movement : once verified the algorithm results in stationary conditions, we would examine how mobility influences the clustering process by controlling both clusters count, shape, and size;

G.3 test the algorithm dynamics when the temperature distribution changes:

That is, these goals reflect the design requirement of supporting sensing/spatial-based clustering in static, mobile, and environment-dynamic scenarios.

4.3 Simulation Framework

We verify our sensing-driven clustering algorithm using simulations. The simulation experiments, resulting data, source code, and instructions for reproducibility are available at a public GitHub repository[6].

Among the many simulators available for swarm-like robots behaviours (e.g. ARGoS (Pinciroli et al, 2012)), we choose Alchemist (Pianini et al, 2013), a meta-simulator for pervasive-computing like applications. Alchemist is already used in similar scenarios (Casadei et al, 2021) and it supports the ScaFi language (Casadei et al, 2020b), that has been chosen among other field-based languages (Viroli et al, 2019) as it supports aggregate processes (Casadei et al, 2019), which we consider essential in order to implement our clustering algorithm.

4.3.1 Parameters

To check the effectiveness of our solution, we evaluate the aggregate program behaviour using different parameters, summarised in Table 2 and described in the following.

One of the most important parameters is the *in cluster threshold* (θ). It defines if a node is inside the cluster or outside; so, it guides the process expansion among the nodes. If the value is too low, the programs take into consideration only a few nodes; if it is too high, the cluster will be expanded to nodes that should not belong to that cluster. This parameter is application-dependent, so developers should carefully choose the right balance between node inclusion and boundedness, ultimately affecting the cluster shape.

The *same cluster threshold* (γ), instead, is used by the cluster leader to define when two clusters are similar (as shown in Section 3.3). This parameter plays a crucial role in finding the right cluster boundaries. Indeed, if γ is too high, two clusters could be merged even if they are different. On the other hand, if γ is too low, multiple overlapped clusters remain even if they could be merged.

A clustering process starts when a node becomes a candidate. *waiting candidate time* (β) rules the rounds needed by a node to spawn a process after it has become a candidate. This helps in avoiding the excessive process spawn due to small local temperature variations.

We are interested in the robustness of the clustering process against the node movement. Therefore, we tested our solution varying the drone *speed* (ω) and the *exploration range* (ζ). We expect that the higher the movement speed, the greater the instability of the identified clusters. ω does not affect candidate nodes, they will stand still until they stay candidates.

We check also how the output changes varying the *density* (α) of drones. Theoretically, we expect a better result with high-density swarms. $N = (10/\alpha)^2$, e.g. with $\alpha = 0.5, N = 400$ and with $\alpha = 0.75, N = 173$.

[6] https://github.com/cric96/experiment-2021-swarm-intelligence-si

Parameter	Unit	Description	Values
In Cluster Threshold – θ	°C	A real value used to verify if the temperature perceived in a certain node could be considered as a part of the current cluster	[0.5, 1.0, 1.5]
Same Cluster Threshold – γ	n.a	A real value used to verify if two clusters could be considered as the same	[0.1, 0.3, 0.7]
Speed – ω	km/s	The constant velocity used by drone to explore the areas	[7, 10, 14]
Exploration range – ζ	km	The maximum range area in which drones could move	[0.5, 0.6]
Density – α	n.a	A parameter used to define how many nodes will be placed in the environment	[0.5, 0.75]
Waiting candidate time – β	n.a	Rounds needed to mark a node as candidate	[3, 5, 7]
Failure frequency – ξ	Hz	Failure frequency of random nodes that participate in the system	[0.5, 0.1, 0]
Spawn frequency – τ	Hz	Spawn frequency of a node in a random position within the environment	[0.5, 0.1, 0]

Table 2: A summary of the parameters used in simulations

4.3.2 Metrics

The clustering results are verified using different metrics. First of all, we extract the number of total unique clusters found by the collective to check if the program produces the correct partitioning. This value gives a quick overview of the clustering result. Along with this value, we evaluate the total number of unique merged clusters. The latter should be as near as possible to the correct cluster number.

However, neither the number of total unique clusters nor the total number of unique merged clusters tells us anything about the shape of the clusters. To this aim, we compute several metrics:

- the number of nodes for each cluster, stating the overall device partitions;
- the Silhouette (Rousseeuw, 1987) and Dunn (Dunn, 1974) indexes, used as internal evaluation schemes;
- the error rate, observable only when we know the ground truth.

By observing the value of the Silhouette index, we can understand if the clusters extracted are overlapped. Indeed, if the Silhouette tends to be 0, it means that the clusters are overlapped. Instead, if it tends to 1, the clusters found are disjointed. The Dunn index, instead, is used as a control value. When we have a Silhouette that tends to be 1, we expect to have a higher Dunn index value.

The error rate metric measures the misclassified nodes: if a node is associated with a cluster but it is far from all the targets in the systems (false positive) or should be associated with a cluster but the algorithm identifies it

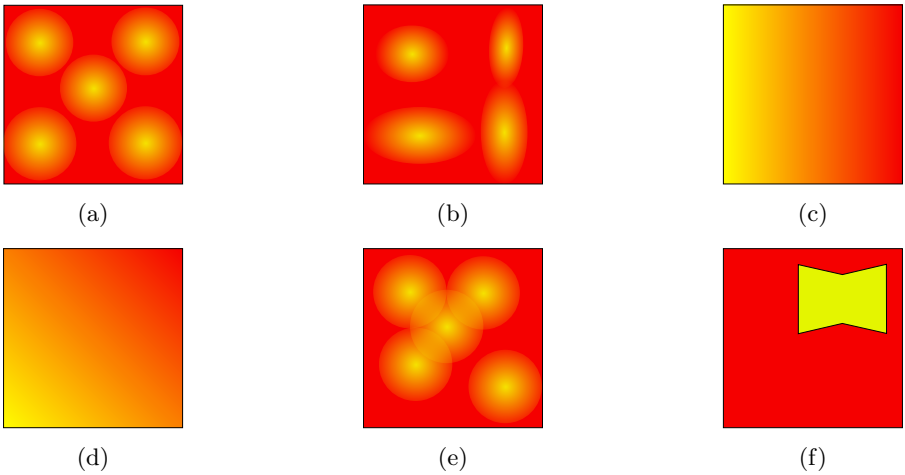


Fig. 3: Graphical representation of temperature field distributions used in the simulations. The lighter the colour, the lower the temperature.

as an external node. The error rate is computed as:

$$E = \frac{FP + FN}{TP + TN}$$

Where TP stand for *true positive* (i.e., number of nodes classified within a cluster and they are placed near to a temperature distribution) and TN stands for *true negative* (i.e., number of nodes classified as external and far from all the temperature distribution) This value is used to understand how well the algorithm performs when the drone explores the areas.

4.4 Simulations

We evaluate the behaviour of our algorithm in several experiments. The simulations have in common i) the environment area (a square with a side of 1km), ii) the communication radius (100 m), and iii) the average evaluation frequency of aggregate programs (1 Hz). The drones are uniformly placed to cover the entire zone. We run the simulations in a modern machine equipped with two AMD EPYC 7301 with 128 GB RAM. The results are reproducible in any modern machine, but consider that it might take a long time to finish (in our configuration, the simulations end after 8 h). Each scenario is executed 20 times with different random seeds for a total of 100 simulated seconds (some simulations lasts 150 s to reach convergence). The data generated by the simulator is handled using NumPy (Harris et al, 2020) and plotted using matplotlib (Hunter, 2007). The plotted results consist of the average (lines)

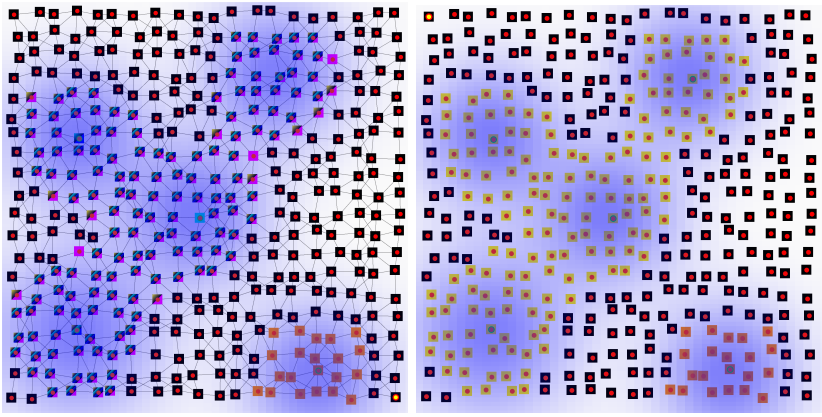


Fig. 4: Snapshots of simulation executions. The colour of the square identifies the cluster id found in that point. Black colour means no cluster. The green circle means that the node is a candidate. The blue gradient circles are a graphical representation of temperature distribution. On the left is shown a snapshot of a simulation before the merge policy has been applied (multiple clusters per point are found). On the right, there is the snapshot of the same simulation after the merge policy action.

and the standard deviation (area behind lines) of the values of interest in different episodes. In Figure 4 there is a graphical representation of a run of our algorithm.

Scenario 1: Gaussian patterns (Figure 3a)

Description. In this scenario, the drones are stationary (i.e., they stand still). There are five zones with a Gaussian distribution, and there is no overlap between distributions. Given the stationary situation, the number of candidate nodes is equal to the number of zones of interest.

Why. Used to verify **G.1**, particularly we expect that the algorithm finds clusters without making any errors and that they will be stable over time.

Scenario 2: Stretched Gaussian patterns (Figure 3b)

Description. These simulations are similar to the previous one, but in this case, the Gaussian distributions have an ellipse-like shape.

Why. With these experiments, we would check that the shape does not make such a difference in the clustering process. Indeed, we expect a result similar to the one in the previous example (**G.1**).

Scenario 3: One direction temperature field (Figures 3c and 3d)

Description. In this case, we imagine that only one cluster is present (fixing θ to 1 °C and putting a total variation of temperature equal to 1 °C). Temperatures grow from left to right in a constant fashion. . we are interested to

see what happens when multiple candidates are elected. In this case, there are several relative minima (the set of nodes that are leftmost with minimum id in their neighbourhood). But, eventually, the processes will expand them in the same way. Thus, we expect that the merging policy tends to create only one cluster. We use the scenario shown in Figure 3d as a reference. Indeed, there will be only one candidate (located in the bottom left corner), and hence the algorithm should result in one cluster.

Why. We devise these experiments to test the effectiveness of the merging policy and to verify the goal **G.1**.

Scenario 4: Gaussian overlapped patterns (Figure 3e)

Description. In this case, we have several Gaussian patterns that could be overlapped. We imagine that the θ value is essential here: if the value is too high, the system will recognise the set of overlapping clusters as one; otherwise, it will consider disjointed.

Why. This experiment serves to emphasize that θ is a domain-dependent choice. Moreover, it will show that the algorithm could be used also to find overlapped situations (**G.1**).

Scenario 5: Non convex patterns (Figure 3f)

Description. In this case, there are two zones, one with a non-convex shape with a lower temperature than the outer zone. Here we expect that, eventually, the system will identify the presence of only two clusters. The program might identify several candidates in the transitory phases (cf. one for each edge). Hence, the merging policy should fix this issue by producing only two clusters.

Why. With this scenario, we want to point out that the program can cope with zones of arbitrary shape.

Scenario 6: Gaussian patterns with movement

Description. We test the result using four Gaussian distributions (arranged similarly to Figure 3a) combined with movement. Here, both merging policy, and *waiting candidate time* (β) will be essential. In particular, β helps to avoid false positives since it waits before spawning a new clustering process when encounters small local temperature variations. In general, we imagine that high values of ω and ζ will make the algorithm more unstable.

Why. We are interested in seeing how movement affects the result of the clustering process (**G.2**).

Scenario 7: Variable size Gaussian pattern

Description. In this experiment, the temperature distributions are placed similarly as Figure 3a, but then the size of areas evolves in time. We expand the areas until a time T and then contract them to their initial size. The starting

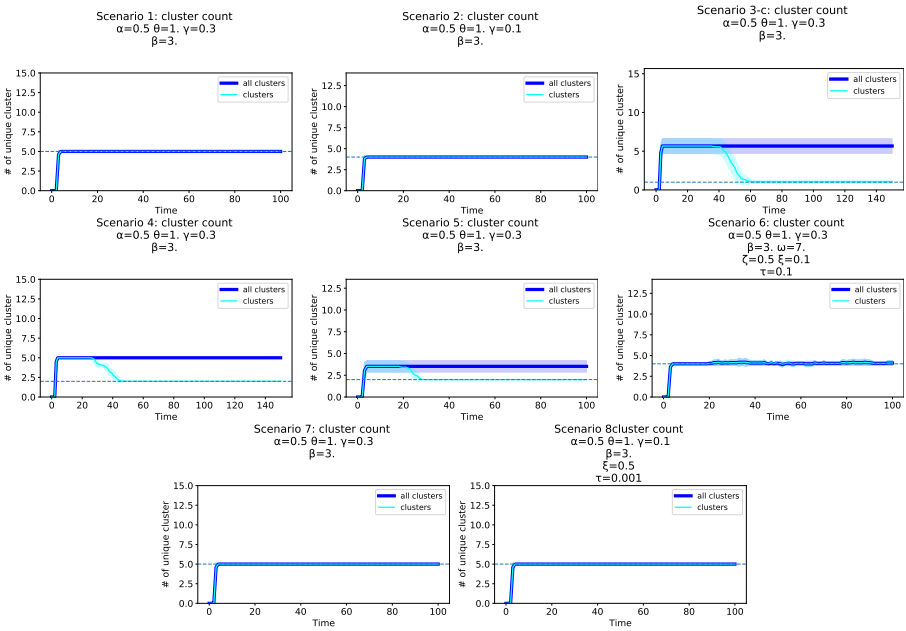


Fig. 5: Overview of simulation results. The dotted lines identify the ideal cluster division count. The blue lines show the unique cluster found. Instead, the cyan lines indicate the unique cluster number after the merging phase.

area range is 100 m, and the maximum area expansion is 1 km. Here we expect that the cluster area follows the underlying temperature distribution.

Why. In this experiment, we verify the algorithm's robustness against temperature changes (**G.3**).

4.5 Results

The simulations underline that the algorithm can find good subdivisions into clusters. Indeed, Figure 5 shows that our algorithm can eventually produce the correct number of clusters after a certain settling period. In the following, we present the result focussing on the evaluation goals stated in Section 4.2.

Goal 1 (G.1): static sensing/spatial-based clustering

Running the simulations of scenarios 1-5 we verified how much the clusters extracted follow the underlying temperature distribution in the static context. Figure 5 shows that the algorithm correctly extracts the cluster number – with the optimal parameters configuration. Furthermore, observing Figure 6, we can deduce that the cluster shape is correct too. Indeed, the Silhouette index tends to be 1 when the clusters are disjointed, and the error rate is negligible.

Here, θ plays a key role. Observing the behaviour of scenario 4 in Figure 7, we see that with too low θ we overestimate the cluster numbers and, with a

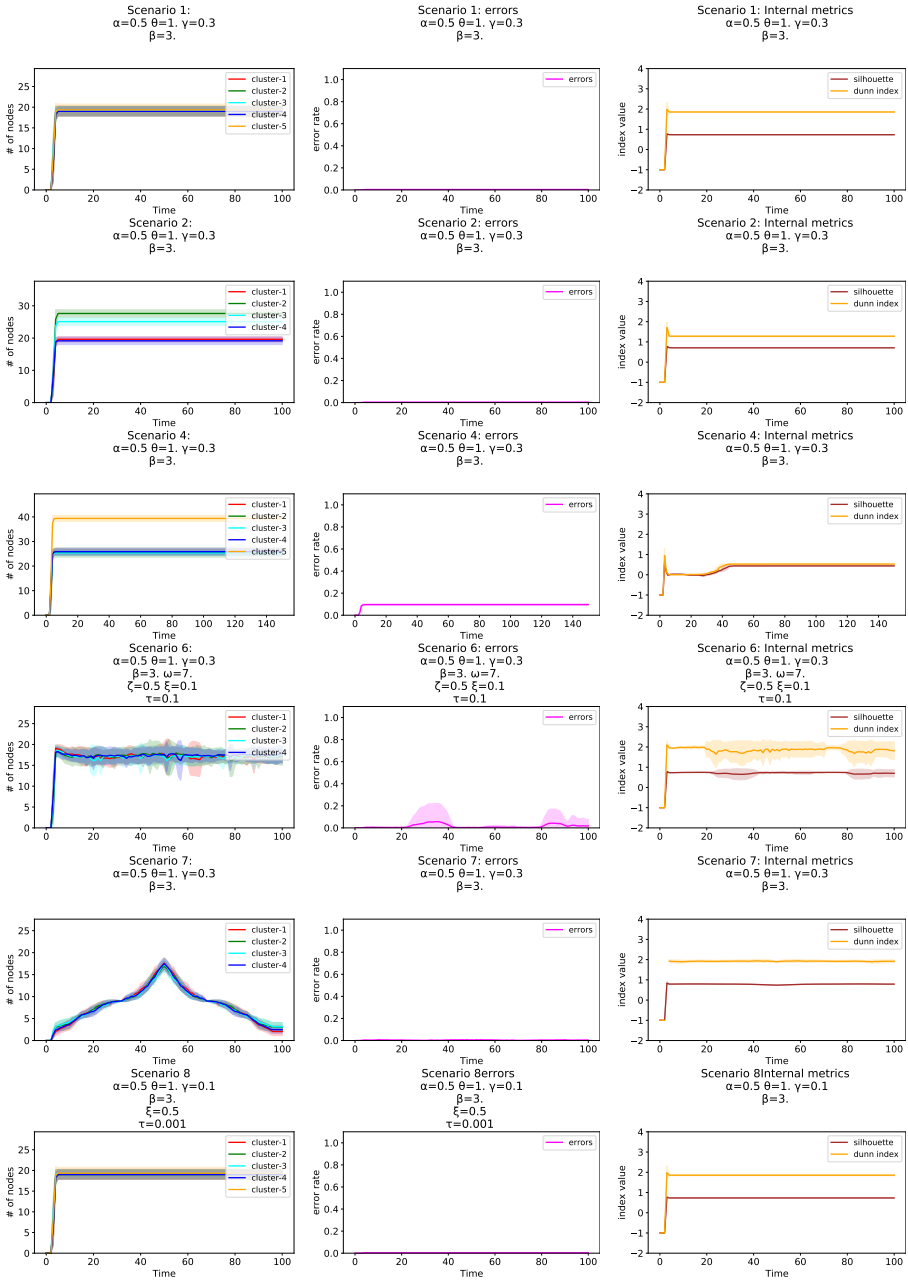


Fig. 6: In-depth analysis of good simulation results. In general, the algorithm produces good results.

high level of θ, we underestimate the cluster number. But this was the expected behaviour, as it depends directly on the trend of the target distributions.

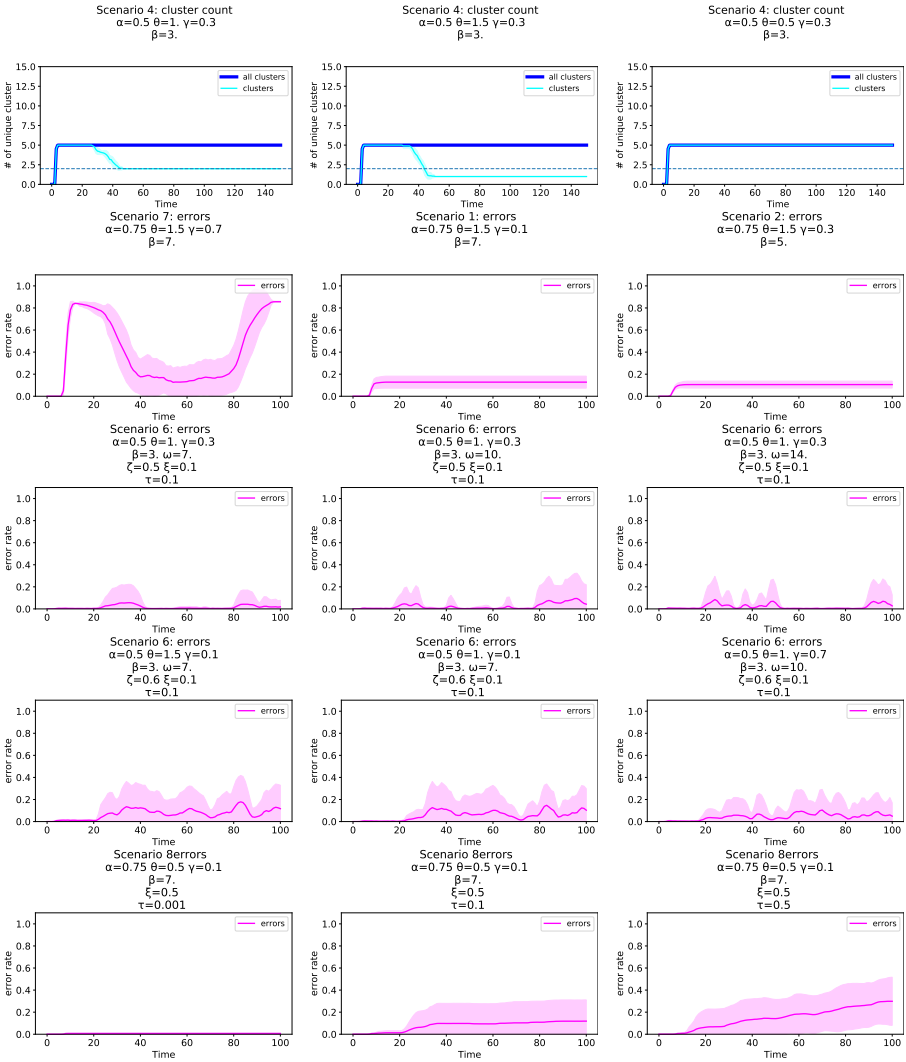


Fig. 7: Main examples of bad clustering results. In the first line, the images show different behaviour varying θ. In the second line, the plots show how the algorithm does not handle well low-density robot swarms. In the third line, the charts show how the algorithm handles various movement speeds. The fourth line shows how the exploration range impacts the clustering results.

Finally, Another important aspect is the density (α) of the system. With a small number of nodes, candidate nodes may be positioned far from the cluster centre, thus identifying wider areas than expected.

***Goal 2 (G.2): robustness against node mobility* and failures**

When nodes have a low mobility and exploration range, the system is robust to node movements (Figure 6). The exploring policy introduces errors, but the results are comparable to solutions where the nodes are stationary. Moreover, even in case of failures, the clustering process is practically not affected at all. However, in the worst case, mobility and failures lead to false positives (Figure 7). Indeed, some processes start in areas where the temperature is almost constant. Therefore, that process approximately covers the whole area (and hence produces a high error rate).

Goal 3 (G.3): robustness against temperature changes

The result of scenario 7 is comparable to the static scenario. Indeed, Figure 6 shows that the cluster number is correct, and Figure 6 shows that the error rate is low and the shape is accurate. The solution suffers from low-density values and wrong θ values as scenarios 1-5 (Figure 7).

4.6 Discussion

Ultimately, our algorithm can support a certain degree of movement, , find various cluster shapes, and cope with temperature changes in the optimal condition: high density (α), limited exploration range (ζ), and an appropriate value for in cluster threshold (θ) value.

However, when drones move randomly, the algorithm starts to produce sub-optimal cluster divisions since the nodes do not care about the cluster found, and they continue to explore the area. But this could lead to becoming a false candidate and then starting an unwanted clustering process. Furthermore, it could be argued that uniform zones are part of a cluster that is not identified as there are no relative minima. For this reason, when a node starts the process in a non-correct zone, the cluster identification will expand in the nearly whole system. This problem could be reduced by changing ω and ζ when the nodes belong to a cluster.

It is worth noting that with a low value of α the algorithm starts to produce bad cluster divisions—. This behaviour is unavoidable since we base our algorithm on the presence of a centroid that starts the clustering process. Indeed, with a low α value, it is more probable that the node that starts the process is far from the real cluster centroid, and hence the process expansion can escape from the underlying distribution, misclassifying a large population of nodes.

5 Related Work

This section covers related work. Coverage of related work is organised to separately cover related clustering models and problems (Section 5.1), research work related to the sensing-based clustering problem we address (Section 5.2), research work related to field-based computing (Section 5.3), and related field-based algorithms (Section 5.4).

5.1 Related Clustering Models and Problems

Clustering is a well-known problem in data analysis and machine learning, and has been widely studied in the literature (Jain et al, 1999; Estivill-Castro, 2002; Jain, 2010). In a classical setting, the data to be clustered is stored in a single dataset, and a single algorithm (or agent) is in charge of finding the "best" clusters according to some optimisation criteria. Each data point in the input data set is described by the values of a fixed set of *features*; the number of such features constitutes the dimensionality of the data set and, typically, high dimensional data is harder to cluster meaningfully.

A characteristic of the clustering tasks considered in the present paper (and in general, of sensing-based methods, see the next section), is that besides the sensed data, a main source of information is the spatial distance between the agents. In (Thrun and Ultsch, 2021), the authors consider high-dimensional data sets that exhibit *natural clusters*, characterised by distances and/or density-based structures. They propose a semi-automated method whereby the clusters are automatically proposed and manually selected starting from a topographic visualisation of the high-dimensional data. Notably, they use swarm intelligence for computing the topographic map, while other techniques are adopted for the interactive process of clusters computation.

There are, however, several works that address swarm-based clustering, using swarm intelligence for the clustering task itself (Martens et al, 2011). It is important to note that such methods (both those based on particle swarm optimisation (PSO), and those based on ant colony systems (ACS)) exploit swarms just as a computational means for finding clusters in a data set. Their goal is not to cluster the elements of the swarm itself, as it is the case for the present work, but to simulate a virtual swarm to find good quality clusters.

Some works directly address the clustering of swarms. In (Hu et al, 2021), the clustering of a team of special robots (i.e., aerial drones) is part of a larger process that, after cluster formation, also involves formation tracking (i.e., tracking a target through a suitable formation), and containment control (i.e., surround ground robots cooperating in the mission). The method proposed to form clusters is based on a game-theoretic framework named GRAPE. A significant difference w.r.t. the present work is that the number (and nature) of clusters is determined by a given set of targets, while we do not assume such a priori knowledge. Another significant work with similar goals is (Ge et al, 2018), where a team of robots must be partitioned into clusters organised as suitable *formations* (i.e., geometric spatial patterns). The proposed solution inter-mixes the determination of clusters and their formation (based, among other things, on the agents dynamics), assuming that the number and nature of such formations is known a priori.

Since we consider clustering over a given topology (network), the problem can be related to graph-based clustering (Chen and Ji, 2010). Graph-based clustering, however, assumes that the given graph can be partitioned into densely connected subgraphs that are sparsely connected to each other; i.e., it assumes that all the similarity information is expressed by the presence of

edges between nodes (and, possibly, by their weights). This is not necessarily the case with the networks formed by our swarms, where connections are just determined by spatial distance, and the clustering is strongly influenced by the sensed data. Also, community detection methods can be viewed as clustering of the nodes of a graph representing a network of relations (e.g. a social network) (Javed et al, 2018). Interestingly, unlike in generic graph-based clustering, communities can easily overlap, since a node (e.g., user) may belong to several communities at once.

5.2 Related Work on Sensing-based Clustering

Sensing-based clustering typically applies to sensor networks that are distributed on a geographical area and exploit clustering mainly to reduce the communication bandwidth and/or energy consumption of the net. The role played by sensing a (possibly dynamic) geographic environment makes such problem and the proposed approaches to solve it relevant to the present work, although the agents considered here are themselves dynamic entities moving and acting across the space.

In (Lin and Megerian, 2007), the goal is to partition sensors for indoor monitoring and control. The cluster heads are predetermined (based on the sources to be monitored and controlled), while cluster formation is periodically scheduled in order to adapt to changes in the sensed data. In our work, instead, the cluster heads are not a priori given: they are determined according to the sensed data (e.g., the agents perceiving local minima) and can change dynamically (e.g., because a candidate withdraws and joins a different cluster).

The goal of (Gedik et al, 2007) is, instead, to obtain energy savings in data collection from a wireless sensor network (WSN) by receiving values from only a subset of selected representatives and predicting the other valuer through automatically generated statistical models. Cluster heads are chosen (probabilistically) based on the amount of energy they have. Cluster formation is periodically scheduled, and the assignment of a sensor to a cluster is based on the distance from the head and the similarity of the sensed value with the head's value. A work with similar goals is (Cai and Zhang, 2018), where again energy savings in a WSN is the primary motivation. Here, the cluster heads are chosen based on residual energy level and data gradient. Moreover, an autoregressive prediction model for sensory data is maintained by each head to self-adjust temporal sampling intervals within the cluster.

A sensing-based clustering problem is also studied in (Kucuk et al, 2020) where, however, instead of being a high energy-constrained WSN, the deployed system involves sensorised units and mobile phones able to upload all the relevant data to the cloud, through cellular and Wi-Fi connections. In a disaster scenario, the mobile phones data is used to centrally compute density-based clusters that can inform the SAR (Search and Rescue) teams about the location of people in the area.

The *DyClee* approach described in (Roa et al, 2019) is also centralised. The authors assume that streams of sensors observations (e.g. in an Industrial

IoT) are continually tracked by their system, and are classified (e.g., as healthy or faulty) based on a set of clusters that capture the patterns corresponding to different states. The main focus is on the novelty detection problem, or concept drift, which implies the ability to update the clusters as new behaviour is learned, while ignoring noise and occasional outliers. The online clustering algorithm consists of two stages based, respectively, on distance and density, and is *fully dynamic* in that it is able to create, eliminate, drift, merge, and split clusters as data is processed.

5.3 Related Approaches and Programming Models

Programming swarms of agents is a difficult task, because of the need of coordinating their local behaviours to achieve global, swarm-level goals. In this work, we adopt the field-based computing and programming approach (Viroli et al, 2019) for expressing self-organising, collective behaviour of swarms. Our focus is on decentralised behaviour-based approaches (rather than automatic design methods like e.g. reinforcement learning), as surveyed e.g. in (Brambilla et al, 2013; Viroli et al, 2019) and briefly in the following.

An approach to the problem that has proven to be quite effective is generative communication through tuple-based coordination models, as offered, e.g., in the Linda language (Gelernter, 1985) and its descendants; essentially, several processes running on the same system can synchronise by writing and retrieving information in a shared (tuple-)space. A derived idea is that of allowing programmability of the tuple space itself, so that the coordination logic of processes can be embedded in the communication medium–see, e.g., (Omicini and Denti, 2001). An obvious limitation of the mentioned approaches for the task of swarm programming is that they assume a central memory accessible by all the agents/processes. However, the idea of tuple-spaces has been extended also to distributed systems, e.g., in the IBM TSpaces framework (Wyckoff et al, 1998).

An important feature of swarm systems is their adaptivity achieved through self-organisation. A support to build such kind of systems is offered by frameworks inspired by other sciences such as biology (Tolksdorf and Menezes, 2003) and chemistry (Sayama, 2009). The field-based computing approach adopted in the present paper is based, instead, on the concept of *field*, borrowed from physics. The related idea of a *field of tuples* has been implemented in the TOTA middleware (Mamei and Zambonelli, 2009).

As seen in this paper, the field-based approach is particularly well suited to mobile, spatially situated agents. A related (and precursor) thread of research of that of *spatial computing*, where space is both an abstraction and a means for computation. Spatial computing approaches have been largely surveyed in (Beal et al, 2013). They are also related to *macro-programming* (Newton et al, 2007), where distributed systems as wholes are programmed by a centralised perspective. For instance, a prominent related macro approach to swarm programming is Buzz (Pinciroli and Beltrame, 2016), where swarms are first-class collection-like abstractions.

5.4 Related Field-based Algorithms

The field-based computing approach adopted in this paper has been applied for programming several swarm intelligence algorithms such as robust distance estimation (*gradient*) (Audrito et al, 2017), leader election (Mo et al, 2018), distributed data collection (Audrito et al, 2021), and team formation and coordination (Casadei et al, 2021).

Field-based computing has the peculiar ability to capture collective behaviours as functions operating on fields and to compose them together as "building blocks" to address problems of increasing complexity (Viroli et al, 2019). Of particular relevance for the present discussion is the implementation of the *SCR (Self-Organising Coordination Regions)* pattern in (Pianini et al, 2021b), where three building blocks are composed to support control and monitoring of a distributed system: the sparse-choice *S block* (used for leader election); the generalised gradient *G block* (for information broadcasting along gradient fields); and the information collection *C block*. Most specifically, the SCR pattern can be denoted as a feedback chain S-G-C-G: leaders are elected (S); then, a gradient from leaders builds the communication structure (G); then, data from members (indirectly defined by the information path towards a leader) is collected towards leaders (C); then, data from leaders is propagated back to the members of the regions (G). However, the SCR pattern is not limited to clustering (S-G part), but also regulates interactions within regions (C-G part). Roughly, the sensing-based clustering algorithm covered in this paper could replace the initial C-G composition that determines the system regions.

Similarly to a clustering algorithm, the S block (Mo et al, 2018) provides a distributed mechanism to elect leaders from a set of candidates, and to assign each remaining *user* node to a leader, thus partitioning the system into regions. The approach presented here is different in several respects: first of all, the candidate leaders are determined by a characteristic of a sensed measure (e.g., local minimum); second, each candidate cluster head spawns an aggregate process to recruit other nodes within the cluster; finally, the other nodes can join more than one cluster, based on the similarity of their sensed values with the ones sensed by leaders.

6 Conclusion and Future Work

In this paper, we precisely define and address the dynamic sensing-based mobile swarm clustering problem. Most specifically, we use the field-based paradigm to develop a novel configurable meta-algorithm promoting self-organised clustering in a swarm of neighbouring-interacting robots. The algorithm is evaluated on a set of synthetic environment configurations. In particular, we show that a swarm can autonomously create clusters reflecting the underlying dynamics of the perceptible target phenomenon in the environment, and is able to deal with a certain degree of change in the swarm topology

and environment. In order to perform the evaluation, we have implemented our algorithms using the ScaFi Scala framework for field-based computing.

Future work could be devised in multiple directions. First of all, it could be interesting to stress and possibly refine the algorithm on more extreme environmental conditions, or to investigate it under different assumptions (e.g., a more constrained or rich system model). Secondly, it could be interesting to compare (or combine) the meta-algorithm against (with) automated swarm behaviour design methods like multi-agent reinforcement learning. Last but not least, we would like to evaluate our algorithms on real use cases, e.g. in smart logistics and precision agriculture scenarios, by implementing them on actual drones or robots.

Declarations

Compliance with Ethical Standards

Conflicts of Interest

The authors declare that they have no conflict of interest.

Funding

Data Availability Statement

The datasets generated during the current study and the simulation framework usable to replicate them are available in a public GitHub repository, https://github.com/cric96/experiment-2021-swarm-intelligence-si.

References

Audrito G (2020) FCPP: an efficient and extensible field calculus framework. In: IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2020, Washington, DC, USA, August 17-21, 2020. IEEE, pp 153–159, https://doi.org/10.1109/ACSOS49614.2020.00037

Audrito G, Casadei R, Damiani F, et al (2017) 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. IEEE Computer Society, pp 91–100, https://doi.org/10.1109/SASO.2017.18

Audrito G, Casadei R, Damiani F, et al (2020) Computation against a neighbour: Addressing large-scale distribution and adaptivity with functional programming and scala. https://doi.org/10.48550/ARXIV.2012.08626, URL https://arxiv.org/abs/2012.08626

Audrito G, Casadei R, Damiani F, et al (2021) Optimal resilient distributed data collection in mobile edge environments. Computers &

Electrical Engineering p 107580. https://doi.org/https://doi.org/10.1016/j.compeleceng.2021.107580, URL https://www.sciencedirect.com/science/article/pii/S0045790621005140

Ball D, Ross P, English A, et al (2013) Robotics for sustainable broad-acre agriculture. In: Alvarez LM, Corke PI, Roberts JM (eds) Field and Service Robotics - Results of the 9th International Conference, December 9-11, 2013, Brisbane, Australia, Springer Tracts in Advanced Robotics, vol 105. Springer, pp 439–453, https://doi.org/10.1007/978-3-319-07488-7_30

Beal J, Dulman S, Usbeck K, et al (2013) Organizing the aggregate: Languages for spatial computing. In: Formal and Practical Aspects of Domain-Specific Languages: Recent Developments. IGI Global, chap 16, p 436–501, https://doi.org/10.4018/978-1-4666-2092-6.ch016

Beal J, Pianini D, Viroli M (2015) Aggregate programming for the internet of things. Computer 48(9):22–30. https://doi.org/10.1109/MC.2015.261

Best G, Faigl J, Fitch R (2018) Online planning for multi-robot active perception with self-organising maps. Auton Robots 42(4):715–738. https://doi.org/10.1007/s10514-017-9691-4

Best G, Cliff OM, Patten T, et al (2019) Dec-mcts: Decentralized planning for multi-robot active perception. Int J Robotics Res 38(2-3). https://doi.org/10.1177/0278364918755924

Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm Intelligence - From Natural to Artificial Systems. Studies in the sciences of complexity, Oxford University Press

Brambilla M, Ferrante E, Birattari M, et al (2013) Swarm robotics: a review from the swarm engineering perspective. Swarm Intell 7(1):1–41. https://doi.org/10.1007/s11721-012-0075-2

Cai W, Zhang M (2018) Spatiotemporal correlation-based adaptive sampling algorithm for clustered wireless sensor networks. Int J Distributed Sens Networks 14(8). https://doi.org/10.1177/1550147718794614

Carrillo-Zapata D, Carranza N, Diego X, et al (2018) Morphogenesis in robot swarms. Sci Robotics 3(25). https://doi.org/10.1126/scirobotics.aau9178

Casadei R, Viroli M, Audrito G, et al (2019) Aggregate processes in field calculus. In: Nielson HR, Tuosto E (eds) Coordination Models and Languages - 21st IFIP WG 6.1 International Conference, COORDINATION 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings, Lecture Notes in Computer Science, vol 11533. Springer,

pp 200–217, https://doi.org/10.1007/978-3-030-22397-7_12

Casadei R, Pianini D, Placuzzi A, et al (2020a) Pulverization in cyber-physical systems: Engineering the self-organizing logic separated from deployment. Future Internet 12(11):203. https://doi.org/10.3390/fi12110203

Casadei R, Viroli M, Audrito G, et al (2020b) Fscafi : A core calculus for collective adaptive systems programming. In: Margaria T, Steffen B (eds) Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part II, Lecture Notes in Computer Science, vol 12477. Springer, pp 344–360, https://doi.org/10.1007/978-3-030-61470-6_21

Casadei R, Viroli M, Audrito G, et al (2021) Engineering collective intelligence at the edge with aggregate processes. Eng Appl Artif Intell 97:104,081. https://doi.org/10.1016/j.engappai.2020.104081

Casadei R, Mariani S, Pianini D, et al (2022a) Space-fluid adaptive sampling: a field-based, self-organising approach. In: ter Beek MH, Sirjani M (eds) Coordination Models and Languages - 24th 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, in press

Casadei R, Pianini D, Viroli M, et al (2022b) Digital twins, virtual devices, and augmentations for self-organising cyber-physical collectives. Applied Sciences 12(1). https://doi.org/10.3390/app12010349, URL https://www.mdpi.com/2076-3417/12/1/349

Chen Z, Ji H (2010) Graph-based clustering for computational linguistics: A survey. In: Proceedings of the 2010 Workshop on Graph-Based Methods for Natural Language Processing. Association for Computational Linguistics, USA, TextGraphs-5, p 1–9, URL https://aclanthology.org/W10-2301/

Clark SS, Beal J, Pal PP (2015) Distributed recovery for enterprise services. In: 2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems, Cambridge, MA, USA, September 21-25, 2015. IEEE Computer Society, pp 111–120, https://doi.org/10.1109/SASO.2015.19

Cruz NB, Nedjah N, de Macedo Mourelle L (2017) Robust distributed spatial clustering for swarm robotic based systems. Appl Soft Comput 57:727–737. https://doi.org/10.1016/j.asoc.2016.06.002

De Masi G, Ferrante E (2020) Quality-dependent adaptation in a swarm of drones for environmental monitoring. In: 2020 Advances in Science and Engineering Technology International Conferences (ASET), pp 1–6, https:

//doi.org/10.1109/ASET48392.2020.9118235

Dolev S (2000) Self-Stabilization. MIT Press

Dunbabin M, Marques L (2012) Robots for environmental monitoring: Significant advancements and applications. IEEE Robotics Autom Mag 19(1):24–39. https://doi.org/10.1109/MRA.2011.2181683

Dunn JC (1974) Well-separated clusters and optimal fuzzy partitions. Journal of cybernetics 4(1):95–104. https://doi.org/10.1080/01969727408546059

Estivill-Castro V (2002) Why so many clustering algorithms: a position paper. SIGKDD Explor 4(1):65–75. https://doi.org/10.1145/568574.568575

Farinelli A, Raeissi MM, Marchi N, et al (2017) Interacting with team oriented plans in multi-robot systems. Auton Agent Multi Agent Syst 31(2):332–361

Garg S, Ayanian N (2014) Persistent monitoring of stochastic spatio-temporal phenomena with a small team of robots. In: Fox D, Kavraki LE, Kurniawati H (eds) Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014, https://doi.org/10.15607/RSS.2014.X.038

Ge X, Han Q, Zhang X (2018) Achieving cluster formation of multi-agent systems under aperiodic sampling and communication delays. IEEE Trans Ind Electron 65(4):3417–3426. https://doi.org/10.1109/TIE.2017.2752148

Gedik B, Liu L, Yu PS (2007) ASAP: an adaptive sampling approach to data collection in sensor networks. IEEE Trans Parallel Distributed Syst 18(12):1766–1783. https://doi.org/10.1109/TPDS.2007.1110

Gelernter D (1985) Generative communication in linda. ACM Trans Program Lang Syst 7(1):80–112. https://doi.org/10.1145/2363.2433

Harris CR, Millman KJ, van der Walt SJ, et al (2020) Array programming with NumPy. Nature 585(7825):357–362. https://doi.org/10.1038/s41586-020-2649-2

Hoshino S (2013) Reactive clustering method for platooning autonomous mobile robots. IFAC Proceedings Volumes 46(10):152–157. https://doi.org/https://doi.org/10.3182/20130626-3-AU-2035.00009, URL https://www.sciencedirect.com/science/article/pii/S1474667015349259, 8th IFAC Symposium on Intelligent Autonomous Vehicles

Hu J, Bhowmick P, Jang I, et al (2021) A decentralized cluster formation containment framework for multirobot systems. IEEE Trans Robotics 37(6):1936–1955. https://doi.org/10.1109/TRO.2021.3071615

Hunter JD (2007) Matplotlib: A 2d graphics environment. Computing in Science & Engineering 9(3):90–95. https://doi.org/10.1109/MCSE.2007.55

Jain AK (2010) Data clustering: 50 years beyond k-means. Pattern Recognition Letters 31(8):651–666. https://doi.org/https://doi.org/10.1016/j.patrec.2009.09.011

Jain AK, Murty MN, Flynn PJ (1999) Data clustering: A review. ACM Comput Surv 31(3):264–323. https://doi.org/10.1145/331499.331504

Javed MA, Younis MS, Latif S, et al (2018) Community detection in networks: A multidisciplinary review. J Netw Comput Appl 108:87–111. https://doi.org/10.1016/j.jnca.2018.02.011

Kemna S, Rogers JG, Nieto-Granda C, et al (2017) Multi-robot coordination through dynamic voronoi partitioning for informative adaptive sampling in communication-constrained environments. In: 2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017. IEEE, pp 2124–2130, https://doi.org/10.1109/ICRA.2017.7989245

Kucuk K, Bayilmis C, Sonmez AF, et al (2020) Crowd sensing aware disaster framework design with iot technologies. J Ambient Intell Humaniz Comput 11(4):1709–1725. https://doi.org/10.1007/s12652-019-01384-1

Lee C, Kim M, Kazadi S (2005) Robot clustering. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Waikoloa, Hawaii, USA, October 10-12, 2005. IEEE, pp 1449–1454, https://doi.org/10.1109/ICSMC.2005.1571350

Lin Y, Megerian S (2007) 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. IEEE, pp 202–206, https://doi.org/10.1109/CCNC.2007.47

Mamei M, Zambonelli F (2009) Programming pervasive and mobile computing applications: The tota approach. ACM Trans on Software Engineering Methodologies 18(4):1–56. https://doi.org/10.1145/1538942.1538945

Mamei M, Zambonelli F, Leonardi L (2004) Co-fields: A physically inspired approach to motion coordination. IEEE Pervasive Comput 3(2):52–61. https://doi.org/10.1109/MPRV.2004.1316820

Martens D, Baesens B, Fawcett T (2011) Editorial survey: swarm intelligence for data mining. Machine Learning 82:1–42. https://doi.org/https://doi.org/10.1007/s10994-010-5216-5

Mo Y, Beal J, Dasgupta S (2018) An aggregate computing approach to self-stabilizing leader election. In: 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Trento, Italy, September 3-7, 2018. IEEE, pp 112–117, https://doi.org/10.1109/FAS-W.2018.00034

Newton R, Morrisett G, Welsh M (2007) The regiment macroprogramming system. In: Abdelzaher TF, Guibas LJ, Welsh M (eds) Proceedings of the 6th International Conference on Information Processing in Sensor Networks, IPSN 2007, Cambridge, Massachusetts, USA, April 25-27, 2007. ACM, pp 489–498, https://doi.org/10.1145/1236360.1236422

Omicini A, Denti E (2001) From tuple spaces to tuple centres. Sci Comput Program 41(3):277–294. https://doi.org/10.1016/S0167-6423(01)00011-9

Pham ND, Le TD, Park K, et al (2010) SCCS: spatiotemporal clustering and compressing schemes for efficient data collection applications in wsns. Int J Commun Syst 23(11):1311–1333. https://doi.org/10.1002/dac.1104

Pianini D, Montagna S, Viroli M (2013) Chemical-oriented simulation of computational systems with ALCHEMIST. J Simulation 7(3):202–215. https://doi.org/10.1057/jos.2012.27

Pianini D, Casadei R, Viroli M, et al (2021a) Time-fluid field-based coordination through programmable distributed schedulers. Logical Methods in Computer Science Volume 17, Issue 4. https://doi.org/10.46298/lmcs-17(4:13)2021

Pianini D, Casadei R, Viroli M, et al (2021b) Partitioned integration and coordination via the self-organising coordination regions pattern. Future Gener Comput Syst 114:44–68. https://doi.org/10.1016/j.future.2020.07.032

Pinciroli C, Beltrame G (2016) Buzz: A programming language for robot swarms. IEEE Softw 33(4):97–100. https://doi.org/10.1109/MS.2016.95

Pinciroli C, Trianni V, O'Grady R, et al (2012) Argos: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intell 6(4):271–295. https://doi.org/10.1007/s11721-012-0072-5

Roa NB, Travé-Massuyès L, Grisales VH (2019) *DyClee*: Dynamic clustering for tracking evolving environments. Pattern Recognit 94:162–186. https://doi.org/10.1016/j.patcog.2019.05.024

Rousseeuw PJ (1987) Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics 20:53–65. https://doi.org/https://doi.org/10.1016/0377-0427(87)90125-7, URL https://www.sciencedirect.com/science/

article/pii/0377042787901257

Saez-Pons J, Alboul L, Penders J, et al (2010) Multi-robot team formation control in the GUARDIANS project. Ind Rob 37(4):372–383

Sayama H (2009) Swarm chemistry. Artif Life 15(1):105–114. https://doi.org/10.1162/artl.2009.15.1.15107

Schranz M, Umlauft M, Sende M, et al (2020) Swarm robotic behaviors and current applications. Frontiers Robotics AI 7:36. https://doi.org/10.3389/frobt.2020.00036

Serugendo GDM, Martin-Flatin JP, Jelasity M, et al (eds) (2007) Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007, Boston, MA, USA, July 9-11, 2007, IEEE Computer Society

Serugendo GDM, Gleizes M, Karageorgos A (eds) (2011) Self-organising Software - From Natural to Artificial Adaptation. Natural Computing Series, Springer, https://doi.org/10.1007/978-3-642-17348-6

Shen W, Will PM, Galstyan A, et al (2004) Hormone-inspired self-organization and distributed control of robotic swarms. Auton Robots 17(1):93–105. https://doi.org/10.1023/B:AURO.0000032940.08116.f1

Testa L, Audrito G, Damiani F, et al (2022) Aggregate processes as distributed adaptive services for the industrial internet of things. Pervasive and Mobile Computing (to appear)

Thrun MC, Ultsch A (2021) Swarm intelligence for self-organized clustering. Artificial Intelligence 290. https://doi.org/https://doi.org/10.1016/j.artint.2020.103237

Tolksdorf R, Menezes R (2003) Using swarm intelligence in linda systems. In: Omicini A, Petta P, Pitt J (eds) Engineering Societies in the Agents World IV, 4th International Workshop, ESAW 2003, London, UK, October 29-31, 2003, Revised Selected and Invited Papers, Lecture Notes in Computer Science, vol 3071. Springer, pp 49–65, https://doi.org/10.1007/978-3-540-25946-6_3

Viroli M, Beal J, Damiani F, et al (2019) From distributed coordination to field calculus and aggregate computing. J Log Algebraic Methods Program 109. https://doi.org/10.1016/j.jlamp.2019.100486

Warren CW (1989) Global path planning using artificial potential fields. In: Proceedings of the 1989 IEEE International Conference on Robotics and Automation, Scottsdale, Arizona, USA, May 14-19, 1989. IEEE Computer

Society, pp 316–321, https://doi.org/10.1109/ROBOT.1989.100007

Wolf TD, Holvoet T (2007) 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. IEEE Computer Society, pp 295–298, https://doi.org/10.1109/SASO.2007.16

Wyckoff P, McLaughry SW, Lehman TJ, et al (1998) T spaces. IBM Systems Journal 37(3):454–474. https://doi.org/10.1147/sj.373.0454