Aggregate centrality measures for IoT-based coordination

Aggregate Centrality Measures for IoT-based Coordination

Giorgio Audrito[a,*], Danilo Pianini[b], Ferruccio Damiani[a], Mirko Viroli[b]

[a] *University of Torino, Italy*
[b] *Alma Mater Studiorum–Università di Bologna, Cesena, Italy*

Abstract

Collecting statistics from graph-based data is an increasingly studied topic in the data mining community. We argue that they can have great value in the coordination of dynamic IoT systems as well, especially to support complex coordination strategies related to distributed situation recognition.

Thanks to a mapping to the field calculus, a distribution coordination model proposed for collective adaptive systems, we show that many existing "centrality measures" for graphs can be naturally turned into field computations that compute the centrality of nodes in a network. Not only this mapping gives evidence that the field coordination is well-suited to accommodate massively parallel computations over graphs, but also it provides a new basic "brick" of coordination which can be used in several contexts, there including improved leader election or network vulnerabilities detection. We validate our findings by simulation, first measuring the ability of the translated algorithm to self-adjust to network changes, then investigating an application of centrality measures for data summarisation.

Keywords: graph algorithms, network centrality, distributed computing, aggregate computing

2010 MSC: 03.5, 05.5, 05.6

*Corresponding author
Email addresses: giorgio.audrito@unito.it (Giorgio Audrito), danilo.pianini@unibo.it (Danilo Pianini), ferruccio.damiani@unito.it (Ferruccio Damiani), mirko.viroli@unibo.it (Mirko Viroli)

1. Introduction

Statistical informations are having an increasingly important role in handling massive graph-based data, such as the Web Graph or social network graphs. In this context, statistical summaries are able to detect local- or global-level features which can give approximate solutions to problems which would not be solvable exactly for graphs of that size. A particularly versatile statistic is the *neighbourhood function* N_G, which calculates the number of vertices within a certain distance from a given vertex, and can be computed efficiently for massive graphs through the state-of-the-art HyperANF algorithm [1]. Through N_G several specific problems can be addressed [2]: among many others, ranking vertices by their *harmonic centrality* (which is easily derivable from N_G).

Vertex ranking is particularly relevant for distributed IoT scenarios, since these measures can help selecting "first-class" vertices (which could fruitfully be promoted to local communication hubs), or weaknesses in the network infrastructure (vertices with few close-range neighbours). Different ranking methods have been developed in various settings, each with its own weaknesses and strengths: for example, *PageRank* is effective for the web graph, but is outperformed by other measures on other kinds of graphs [1]. Algorithms computing those metrics need to satisfy several non-trivial requirements in order to be suitable for fully-distributed IoT scenarios, such as: resilience to network changes, continuous adaptation to input evolution (formalizable by *self-stabilisation* [3]), near-constant time complexity for each computational unit, locality of interactions (obtainable with the *aggregate programming* paradigm [4] for designing individual behaviour from a collective viewpoint). Porting algorithms to the IoT is thus a non-trivial task which may require extensive work where natural translations are not possible or sufficiently performing.

In this paper, we provide a natural translation of four centrality measures (degree, PageRank, closeness, harmonic) in *field calculus* [5], a tiny functional language for expressing "aggregate" coordination policies: essentially, a field calculus program specifies the dynamic behaviour of a "computational field"

([6, 7, 8]), a map from nodes of the network to computational values. While the first two measures are translated directly from their definition, the last two are obtained from the HyperANF algorithm [1], hence providing an efficient approximation of the neighbourhood function. Even though the translation is a natural porting to field calculus, it automatically empowers the algorithms with the ability to scale over arbitrarily large distributed networks, and to react to transient changes or faults. This latter property can be formalised through the notion of *self-stabilisation*, which can be proved for the translated algorithms (except for PageRank) thanks to the results in [3]. To the best of our knowledge, self-stabilisation for algorithms computing centrality measures, hence their computation on mutable inputs, have not been investigated in previous works.

As key contribution, we then show that such implementation of centrality measures can be applied to solve important coordination problems in IoT-based systems: specifically, in this paper we show it can be used to improve selection of local coordinators to support situation recognition tasks [9, 10, 11]. More generally, we believe this translation gives an example suggesting that field calculus is well-suited for expressing massively parallel computations over graphs, and paves the way towards more extensive application of graph-based algorithms to IoT systems. The translation of the HyperANF algorithm (in Section 3.3) in field calculus has been already presented in [12] (without experimental tests and self-stabilisation checks); all the other results presented in this paper are new. Section 2 introduces the four centrality measures and the other concepts needed to express the translation. Section 3 provides the translation, together with a motivation outlining its possible use cases. Section 4 evaluates the centrality measures in two simulation scenarios: the first measures the error of harmonic centrality estimations in a mutable scenario; the second compares the effectiveness of various centrality measures in guiding the selection of a central coordinator. Section 5 discusses related work and Section 6 summarizes the paper and outlines directions for future work.

P	::=	$\overline{\texttt{F}}\ \texttt{e}$	program
F	::=	$\texttt{def d}(\overline{\texttt{x}})\ \{\texttt{e}\}$	function declaration
e	::=	$\texttt{x} \mid \texttt{v} \mid \texttt{e}(\overline{\texttt{e}}) \mid \texttt{if}(\texttt{e}_0)\{\texttt{e}_1\}\ \texttt{else}\ \{\texttt{e}_2\}$	expression
		$\mid \texttt{nbr}\{\texttt{e}\} \mid \texttt{rep}(\texttt{e})\{(\texttt{x})\texttt{=>e}\} \mid \texttt{share}(\texttt{e})\{(\texttt{x})\texttt{=>e}\}$	
v	::=	$\phi \mid \ell$	value
ϕ	::=	$\overline{\delta} \mapsto \overline{\ell}$	neighbouring field value
ℓ	::=	$\texttt{c}(\overline{\ell}) \mid \texttt{f}$	local value
f	::=	$\texttt{b} \mid \texttt{d} \mid (\overline{\texttt{x}})\texttt{=>e}$	function value

Figure 1: Syntax of Field Calculus.

2. Background

2.1. Field Calculus

The field calculus is a tiny functional language for formally and practically expressing aggregate programs: a detailed account of it is given in [5, 3, 13]—we hereby recollect its most basic characteristics in order to be able to use it as common language to express the algorithms used in this paper with actual executable code.

The syntax of field calculus is given in Figure 1. In field calculus a program $\texttt{P}$ consists of a sequence of function declarations $\overline{\texttt{F}}$ and of a main expression $\texttt{e}$—following [14], the overbar notation denotes metavariables over sequences: e.g., $\overline{\texttt{e}}$ ranges over sequences of expressions $\texttt{e}_1, \ldots, \texttt{e}_n$ with $n \geq 0$. On each device δ the expression $\texttt{e}$ evaluates to a value $\texttt{v}$ that may depend on the state of δ (values sensed by δ, the result of previous evaluation, and information coming from neighbours). Therefore the expression $\texttt{e}$ induces a *computational field* Φ, which can be represented as a time-varying map $\delta_1 \mapsto \texttt{v}_1, \ldots, \delta_n \mapsto \texttt{v}_n$, assigning a value $\texttt{v}_i$ to each device δ_i in a network. Each device δ updates its value (by evaluating $\texttt{e}$) in asynchronous computational rounds.

The syntax of an expression $\texttt{e}$ comprises several constructs, which we briefly present. Firstly, $\texttt{e}$ can be a variable $\texttt{x}$, used as function formal parameter, or a

value `v` which in turn could be either a *neighbouring field value* ϕ (associating to each device a map from neighbours to local values), or a *local value* ℓ (built-in functions `b`, user-defined functions `d`, anonymous functions $(\overline{\texttt{x}})$`=>e`, or built-in values $\texttt{c}(\overline{\ell})$ where `c` is a data constructor). Secondly, `e` can be a function call $\texttt{e}(\overline{\texttt{e}})$ where an expression of functional type `e` is applied by-value to arguments $\texttt{e}_1, \ldots \texttt{e}_n$; or an `if`-expression $\texttt{if}(\texttt{e}_0)\{\texttt{e}_1\}\,\texttt{else}\,\{\texttt{e}_2\}$ modelling domain restriction, which computes $\texttt{e}_1$ in the devices where $\texttt{e}_0$ is true, and $\texttt{e}_2$ on the others.

Finally, `e` can be a `nbr`-, `rep`- or `share`-expression, which are the three most characteristic constructs of field calculus. An `nbr`-expression $\texttt{nbr}\{\texttt{e}\}$ models neighbourhood observation, by producing a neighbouring field value that represents an "observation map" of neighbour's values for expression `e`, namely, associating to each device δ (that has evaluated $\texttt{nbr}\{\texttt{e}\}$ during its last update) a map from neighbours to their latest evaluation of `e`. A `rep`-expression $\texttt{e} = \texttt{rep}(\texttt{e}_1)\{(\texttt{x})\texttt{=>e}_2\}$ models time evolution and state preservation, by repeatedly applying the anonymous function $(\texttt{x})\texttt{=>e}_2$ to the value computed for `e` in the previous round, starting from $\texttt{e}_1$ if no previous value is available. A `share`-expression $\texttt{e} = \texttt{share}(\texttt{e}_1)\{(\texttt{x})\texttt{=>e}_2\}$ combines aspects of both `rep` and `nbr`: it models state preservation, evolution and sharing, by repeatedly applying $(\texttt{x})\texttt{=>e}_2$ to the *neighbouring field value* of the values computed for `e` *in the last round of neighbour devices* (including the previous round of the same device).

Example 1. In order to showcase a field calculus program at work, consider function `ever` below, computing whether a certain distributed Boolean value `v` has ever became false in some device of the network.

```
1  def ever(v) {
2    share (false) { (x) => or(v, any(x)) }
3  }
```

The above program can be understood as evolving a distributed Boolean value `ever(v)`, by repeatedly updating it in each computational round of the network. In every device, this value is first initialised to `false`, and then updated by

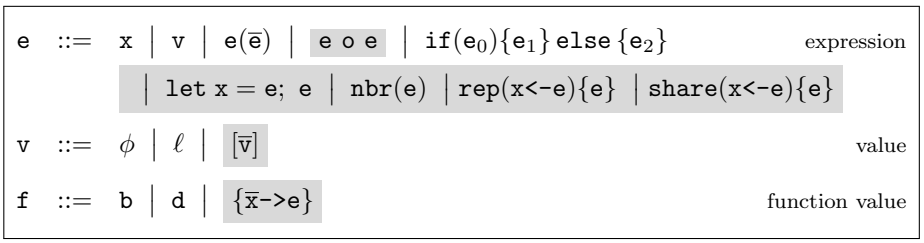

Figure 2: Syntax of the fragment of Protelis used in this paper. Differences with Fig. 1 are highlighted in grey, and identical production rules are omitted.

gathering the neighbours' values of `ever(v)` as a neighbouring field value into `x` (including the value of `ever(v)` in the previous round on the same device, if it exists). This value is first collapsed to a single Boolean value through `any(x)`, which is true if `x` (hence `ever(v)` is true) is true for any neighbour. Then, the current value of `ever(v)` is obtained by combining the result of `any(x)` with `v`, becoming true in case `v` is true in the current round. This final result is then implicitly shared with neighbours for their following computational rounds.

2.2. Protelis

Protelis [15] is a Java-interoperable[1] domain-specific language, implemented via Xtext [16, 17], which reifies the field calculus semantics. Its syntax has been designed to be familiar to those acquainted Java and Kotlin, and its runtime requires a Java Virtual Machine. Protelis has been used for several distributed scenarios, including among others: crowd steering [18], adaptive service dispersal [19], and airborne sensor sharing [20]. All the algorithms presented in this paper have been implemented in Protelis for evaluation; their complete implementation is available online, and essential parts of it are reported in the paper.

Figure 2 presents the fragment of the Protelis language used in the code snippets presented in this paper. Most of the syntax is identical to that of field

[1] The interoperation capabilities are actually targeting JVM types regardless of the language which generated them: Protelis can de fact interoperate also with Kotlin, Scala, and other JVM languages.

calculus in Figure 1. Thus, identical production rules are omitted and differences are highlighted in grey. Three additional constructs are present:[2] application of binary operators `o` in infix notation, let-binding expressions `let x` $= \mathtt{e}_1; \mathtt{e}_2$, and the notation $[\mathtt{e}_1, \ldots, \mathtt{e}_n]$ for tuple construction. The syntax of the three characteristic field calculus constructs is slightly modified: `nbr` uses parentheses instead of curly braces, `rep` and `share` have the variable declaration together with the initialisation expression instead of with the update expression. Finally, the syntax of anonymous functions is modified to follow that of the Kotlin language, where the whole expression is between curly braces. Overall, these modifications are essentially cosmetic: for example, in Protelis function `ever` in Example 1 can be rewritten as follows.

```
def ever(v) {
  share (x <- false) { v || any(x) }
}
```

2.3. Centrality Measures

Many different techniques for collecting statistics from graph-based data are being studied in the data mining community, each able to address different questions. In this paper, we focus on statistics measuring centralities of the nodes of the graph. Among the many ones ever proposed, in particular, we focus on the most common ones, allowing for efficient distributed computation, listed in the remainder of this section.

2.3.1. Degree centrality

The historically first and conceptually simplest centrality measure is the so-called *degree centrality*, which is defined as the number of links incident upon a node (i.e. its degree). Intuitively, nodes with an higher number of links are less prone to encounter network disconnections, hence may be elected as communication hubs more effectively than nodes with a lower degree. Athough this

[2] In field calculus, all those constructs can be defined as syntactic sugar in terms of the other existing constructs.

measure is particularly simple and efficient to calculate, compared with other centrality measures, degree centrality is usually the least effective: in approximately homogeneous situated networks, nodes at the edge of the network will have lower degrees, while all other nodes will have similar degrees, hence selecting the node with the highest degree mostly accounts to selecting a random node which is not at the network edge.

2.3.2. PageRank

A very popular centrality measure in the data mining community is *PageRank* [21], first introduced for the Google search engine, which is an instance of the broader class of *eigenvector centrality measures*. According to this measure, the centrality score r_i of a node i is the fixed point of the system of equations:

$$r_i = (1 - \alpha) + \alpha \sum_{j \in \text{neigh}(i)} \frac{r_j}{\deg(j)}$$

where α is a parameter (usually set at 0.85 [22]), $\deg(j)$ is the degree of node j and $\text{neigh}(i)$ is the set of neighbour nodes j connected to i. This centrality measure have been proved effective on logical graphs such as the web graph, and can be efficiently calculated by re-iterating the equations above for each node, starting from $r_i^0 = 1$. However, it is usually not very effective on approximately homogeneous situated networks: in this case, PageRank favours nodes with many neighbours with a lower degree, which usually are nodes almost at the edge of the network. In fact, doubts on its effectiveness have also been risen on logical graphs [1] with respect to distance-based measures (in particular harmonic centrality), and will be experimentally proven in section 4.3.

2.3.3. Neighbourhood function

The most effective centrality measures considered [1] are *harmonic* and *closeness* centrality, both derivable from (variations of) the *neighbourhood function* of a graph.

Definition 1 (Neighbourhood Function)**.** Let $G = \langle V, E \rangle$ be a graph with n vertices and m edges. The *generalized individual neighbourhood function*

$N_G(v, h, C)$, given $v \in V$, $h \geq 0$ and $C \subseteq V$, counts the number of vertices $u \in C$ which lie within distance h from v. In formulas, $N_G(v, h, C) = |\{u \in C : \ \mathrm{dist}(v, u) \leq h\}|$.

Elaborations of the N_G values have been used to answer many different questions [1, 2], including: graph similarity, vertex ranking, robustness monitoring, network classification. Since exact computation of N_G is impractical, requiring $O(nm)$ time in linear memory and $O(n^{2.38})$ time in quadratic memory, effort has been spent in developing fast algorithms approximating N_G up to a desired precision.

In Palmer et al. [2], the problem of computing N_G is reduced to that of succinctly maintaining size estimates for sets upon set unions. Define $\mathcal{M}_G(v, h, C)$ as the set of vertices in C within distance h from v, so that $N_G(v, h, C) = |\mathcal{M}_G(v, h, C)|$. Notice that $\mathcal{M}_G$ can be computed recursively as:

$$\mathcal{M}_G(v, h, C) = \bigcup_{(vu) \in E} \mathcal{M}_G(u, h - 1, C)$$

where $\mathcal{M}_G(v, 0, C) = \{v\} \cap C$. If we represent the sets $\mathcal{M}_G(v, h, C)$ through succinct counters, the previous formula translates into an algorithm computing estimations of N_G. Vigna et al. [1] later improved the original algorithm by using a more effective class of estimators, the *HyperLogLog counters* [23], by expressing the "counter unions" through a minimal number of broadword operations, and by engineering refined parallelisation strategies. HyperLogLog counters maintain size estimates with asymptotic relative standard deviation $\sigma/\mu \leq 1.06/\sqrt{k}$, where k is a parameter, in $(1 + o(1)) \cdot k \cdot \log\log(n/k)$ bits of space. Updates are carried out through k independent "max" operations on $\log\log(n/k)$-sized words. It follows that N_G, given a fixed precision, can be computed in $O(nh)$ time and $O(n \log\log n)$ memory, allowing it to be applied on very large graphs such as the Facebook graph [24].

2.3.4. Closeness centrality

Closeness centrality c_i of a node i is defined as the reciprocal of the total distance to other nodes, and can be computed in terms of the neighbourhood

function as in the following equation:

$$\frac{1}{c_i} = \sum_{j \neq i} \text{dist}(i,j) = \sum_{h=1}^{D} h \left(N_G(i,h,V) - N_G(i,h-1,V) \right)$$

where D is the graph diameter (maximum distance between nodes in G).

2.3.5. Harmonic centrality

Harmonic centrality h_i of a node i is defined as the sum of the reciprocals of distances to other nodes, and can also be computed in terms of the neighbourhood function as in the following equation:

$$h_i = \sum_{j \neq i} \frac{1}{\text{dist}(i,j)} = \sum_{h=1}^{D} \frac{N_G(i,h,V) - N_G(i,h-1,V)}{h}$$

where D is the graph diameter (maximum distance between nodes in G). Vertices with high harmonic (or closeness) centrality are best-suited to be elected as leaders for coordination mechanisms, since they are connected to many other vertices through a small number of hops.

3. Aggregate Graph Statistics

We now show how all centrality measures and statistics in section 2.3 can be naturally translated into field calculus. This transformation suggests that the field calculus is a natural framework for parallel computations, so that a future cloud-based implementation of it (as envisaged in [25]) could be used to address a relevant class of "traditional" massively parallel computations. Furthermore, this translation has additional relevance in the aggregate computing context, since the resultant algorithms are able self-adjust to dynamic changes in inputs (i.e., they have the *self-stabilisation* property and belong to the *self-stabilising fragment* of the calculus [3]). In fact, these algorithms altogether provide a new *centrality building block* which can be used to produce various dynamic vertex rankings, able to classify vertices by features of their neighbourhoods. These rankings can in turn be used either to recognise network vulnerabilities (vertices with few short-range neighbours and many long-range neighbours), or

to elect leaders for other coordination mechanisms (vertices with high degrees of "centrality").

3.1. Degree centrality

Degree centrality can be easily written in field calculus (in particular in its implementation Protelis [15]) through the following one-liner, computing it by: *(i)* creating a constant field whose value is 1 everywhere, and *(ii)* summing the contribution of every neighbour, hence counting them.

```
def degree() { foldSum(0, nbr(1)) }
```

This function belongs to the self-stabilising fragment identified in [3], since it does not contain any occurrence of `rep` or `share`.

3.2. PageRank

PageRank centrality can be easily written in Protelis as the following simple program:

```
def pagerank() {
  share (rank <- 1) {
    foldSum(0.15, 0.85 / nbr(max(degree(), 1)) * rank)
} }
```

Here, the rank is repeatedly updated by adding 0.15 with the sum of neighbours' ranks divided by their degree (and multiplied by 0.85). Unfortunately, this function does not belong to the self-stabilising fragment identified in [3]. Experimental evaluation in Section 4 suggests that this function is indeed self-stabilising, although a formal proof of this property is left as future work.

3.3. Neighbourhood function and HyperANF

Assume that we are given an `HyperLogLog` type, which can be constructed out of numerical elements, composed through a union operator, and inspected for size through member function `getCardinality`. Moreover, consider a function `myself`, returning a `HyperLogLog` built using solely the local node unique identifier. We can then express HyperANF through the following simple field calculus code,

computing a list of floating-point estimations $\langle N_G(\delta, i, C) : \ i = 0 \dots h \rangle$ in each device δ, provided that `source` is true if and only if $\delta \in C$.

```
def hyperANF(depth, source) { // Recursive implementation of hyperANF
  if (depth == 0) { // End of the recursion, the result is a HLL with myself
    [myself(), [myself().getCardinality()]]
  } else {
    // Recurse one level and get the result
    let recursion = hyperANF(depth - 1, source);
    // Compute the union of the local and neighbours' results at this depth
    let union = foldUnion(nbr(recursion.get(0)));
    // Return both the union of results and the list of cardinalities per-depth
    [union, recursion.get(1).append(union.getCardinality())]
  }
}
```

Note that the above function belongs to the self-stabilising fragment identified in [3], since it does not contain any occurrence of `rep` or `share`.

In Vigna et al. [1], non-trivial effort was made to allow for efficient parallel computation of HyperANF, since the algorithm did not fit inside existing parallel computation frameworks (e.g., MapReduce [26]). Thus, it is worth noting that the field calculus is instead able to easily capture the algorithm, allowing for an "automatic" parallelisation and naturally extending the original algorithm to an "on-line" context (i.e., self-adapting to changes of either network structure or source vertices).

3.4. Harmonic and closeness centrality

The result provided by `hyperANF` can be used to implement real-time vertex ranking calculations for efficient leader election or network weaknesses detection. In the following Kotlin code, we succinctly express the computation of harmonic centrality and closeness centrality based on the estimates produced by `hyperANF`. We picked Kotlin instead of Protelis for this part of the imple-

mentation[3] for the following reasons: *(i)* once `hyperANF` has been executed, the information is entirely available and the remainder of the computation can be local; *(ii)* Protelis is designed for succinctly express field computations, but its standard library is less expressive and rich than Kotlin's; *(iii)* which in turn makes the Kotlin implementation much more compact and clean.

```kotlin
// Extension function representing a mapping over a sequence of HHL counters
fun <R> Sequence<HyperLogLog>.indexedMap(
    operation: (IndexedValue<Long>) -> R // mapping operation on cardinalities
): Sequence<R> = map { it.cardinality } // map HLL to its cardinality
    // type is now Sequence<Long>
    .zipWithNext { a, b -> b - a } // subtract the previous cardinality
    .withIndex() // provide distance information as index
    // Type is now IndexedValue<Long>, namely a pair of index and cardinality
    .map(operation) // run the mapping

// Divide each cardinality for the distance, then sum
fun harmonicCentrality(cardinalities: Iterable<HyperLogLog>): Double =
    cardinalities.asSequence()
        // it.value is the cardinality, it.index the recursion depth
        .indexedMap { it.value.toDouble() / (it.index + 1) }.sum()

// Multiplicative inverse of the sum of the product of cardinalities
// with their respective distances
fun closenessCentrality(cardinalities: Iterable<HyperLogLog>): Double =
    1 / cardinalities.asSequence()
        // it.value is the cardinality, it.index the recursion depth
        .indexedMap { it.value.toDouble() * (it.index + 1) }.sum()
```

Extension function `Sequence<HyperLogLog>.indexedMap` is a higher order transformation operation which prepares the `Sequence<HyperLogLog>` to be used in harmonic and closeness centrality computation by computing the cardinality of each recursion level. The function `operation: (IndexedValue<Long>) -> R` taken as parameter is responsible to reduce the couples of (index, cardinality) to the

[3]Indeed this part of the software is actually written in Kotlin, it's not just for a cleaner presentation in the manuscript, see `https://archive.is/npvCe`

desired final value. `Sequence<HyperLogLog>.indexedMap` factorises the common behaviour of `harmonicCentrality` and `closenessCentrality`, allowing for a simplified implementation.

3.5. Leader election

As a further motivating example, we can use any of the previously introduced centrality measures to elect a leader for coordination routines. Given (an upper bound of) the network diameter, the node with highest centrality can be selected through the following routine:

```
def leaderElection(rank, id, diameter) {
  // simmetry is broken by using rank. The second parameter of
  // the following triple is the distance from the closest leader
  // the third parameter is the id of the leader (candidate)
  let local = [-rank, 0, id]; // the default candidate is myself
  share (neigh <- local) { // compute the field or neighboring candidates
    // discard candidates that are farther than the diameter upper bound
    let options = mux (neigh.get(1) <= diameter) { neigh } else { local };
    let best = foldMin(local, options); // pick the best candidate
    // I'm one hop more distant from the leader than my neighbour
    best.set(1, best.get(1) + 1)
  }.get(2) // return the id of the leader
}
```

The routine logic is the following: *(i)* a candidature is expressed as a triple `[rank, id, distance]`; *(ii)* by default, each node candidates itself; *(iii)* candidates from the neighbourhood are retrieved; *(iv)* candidates whose distance is farther than the upper bound of the network diameter left the network and thus get discarded; *(v)* among the remaining candidates, the one with the best rank is selected; *(vi)* in case of ties, the closest one is selected, and in case of further ties the one with the lowest id is selected; *(vii)* the distance from the leader is the distance from the leader to the neighbour plus one extra hop.

4. Evaluation

We evaluate our proposed approach via simulation in three phases: first, we investigate the approximation quality of the proposed approach by measuring the error of estimating the harmonic centrality for a network of devices via `hyperANF` with respect to an oracle; second, we show how the proposed approach allows for a self-stabilising and completely distributed leader election, and we proposed the technique to dynamically create fog colonies [27]; third, we provide evidence that relying on aggregate graph statistics can be valuable for higher level aggregate computing applications.

Aggregate code has been written in Protelis [15], and simulations have been performed using Alchemist [28]. Alchemist[4] is a flexible simulator based on an extended version of the Gibson-Bruck kinetic Monte Carlo algorithm [29], used in several scientific publications for rather diverse scenarios, ranging from smart cities [4] to drone swarms [30] to morphogenesis [31]. Alchemist provides native integration with various aggregate computing languages, such as Scafi [32] and Protelis [15], and is equipped with a batch engine for automating the execution of several repetitions of a simulation; making it a straightforward choice for the analysis to be presented. Generated data has been processed using xarray [33]; visual reports of have been created via matplotlib [34]. For the sake of reproducibility, the experiment code is available online[5], along with instructions, a fully automated execution system, and continuous integration in place[6]. The entire data analysis includes over 1000 charts, available on the aforementioned source. For the sake of brevity, only the most relevant data is discussed in this manuscript.

4.1. Quality of the Estimates

For the first experiment, we deploy a network of n mobile nodes in a square arena with a 500m wide side. Nodes are free to move within the arena using

[4] `alchemistsimulator.github.io`

[5] `https://github.com/DanySK/Experiment-2020-SCP-Graph-Statistics`

[6] `https://travis-ci.org/github/DanySK/Experiment-2020-SCP-Graph-Statistics`

Name	Description	Values	Unit
$\|\vec{v}\|$	node speed	0, 1, 2, 5, 10	$\frac{m}{s}$
n	device count	$\{1, 2, 5\} \times \{10, 10^2, 10^3\}$	devices
$\mathbf{E}[N]$	mean neighbours	2, 3, 5, 7, 10	devices

Table 1: List of the variables and their values for the simulations.

Lévy walks [35] with velocity $\vec{v}$ and constant speed $|\vec{v}|$, bouncing off the arena boundary. Nodes communicate with devices located at a distance shorter than $500\sqrt{2\pi N}$ meters; hence, they communicate on average with $\mathbf{E}[N]$ other devices. Every node x estimates its normalized[7] harmonic centrality $H_a(x)$ in asynchronous fairly scheduled rounds with a frequency of 1Hz. This value is compared with the exact harmonic centrality value for node x, $H(x)$, as provided by an oracle. The local absolute error is evaluated as $e_x = |H_a(X) - H(x)|$, and the mean absolute error is computed for the whole network as $MAE = \frac{1}{n}\sum_{x=1}^{n} e_n$.

Each combination of variable values (namely, for each member of the set representing the Cartesian product of the possible values of each variable in table 1) is tested 20 times changing the simulation seed, which in turn modifies the initial node deployment, the movement of nodes, and the round scheduling times.

Data shows that, as expected, larger networks require a longer time to converge, as pictured in fig. 3, fig. 4, fig. 5, and fig. 8. This is due to the stabilisation time of the algorithm scaling linearly with the network diameter: the larger the diameter, the longer the stabilisation time. Denser networks feature a higher initial error peak, but they then converge to the final value more quickly than sparser networks (again, due to shorter diameter) as depicted in figs. 3 and 7. Intermediate densities achieve the worst results in our tests (see figs. 5 to 7).

[7]The formulas given in section 2.3 for harmonic and closeness centrality produce values which are larger for larger networks, and are usually normalised by dividing for the total network size.

Intuitively, higher values for N are expected to produce networks with a single large connected component, while lower values are expected to produce networks with multiple small connected components. This intuition is formally captured by the concept of percolation threshold: above such threshold, there exist a single large connected component encompassing the whole system. The percolation threshold of a disk communication model has being measured at $N = 4.5123495$ neighbours [36]: thus, an average of less than five neighbours causes frequent partitioning on the network graph into sub-networks, which quickly converge to their final value. Instead, an average of five neighbours causes the computation to be performed on a large but sparse network, hence increasing error and volatility. Error introduced by node mobility stacks with the error due to the natural time required for algorithm stabilisation. Initially, the contribution of the latter error is dominant, but it then progressively shrinks with the algorithm stabilisation. Error on the later phases of the simulation are instead mostly due to the disruption induced by node mobility and subsequent network reconfiguration. This kind of two-phased error dominance give rise to the local minimum in error seen in fig. 4 and fig. 8, as well as the behaviour observed in fig. 6, fig. 7.

4.2. Application: fog colonies partitioning

Measures of node centrality can be used in processes involving leader election to determine a set of good candidates, or the leader itself. One possible application of a leader election process is partitioning of a network into colonies, each containing a single leader node which can communicate with other leaders. This sort of network overlay has been used in several works found in literature [27, 37, 38, 39], and has a wide range of applications, including vehicular networks [40], artificial biology [41], coordination of swarms of construction robots [42], and edge systems resource management and monitoring [43, 44, 45].

In this section, we excercise our distributed version of HyperANF in a setup similar to the one presented in [27], with the goal of showing how a different leader selection policy may affect the overlay network shape. To this end, we

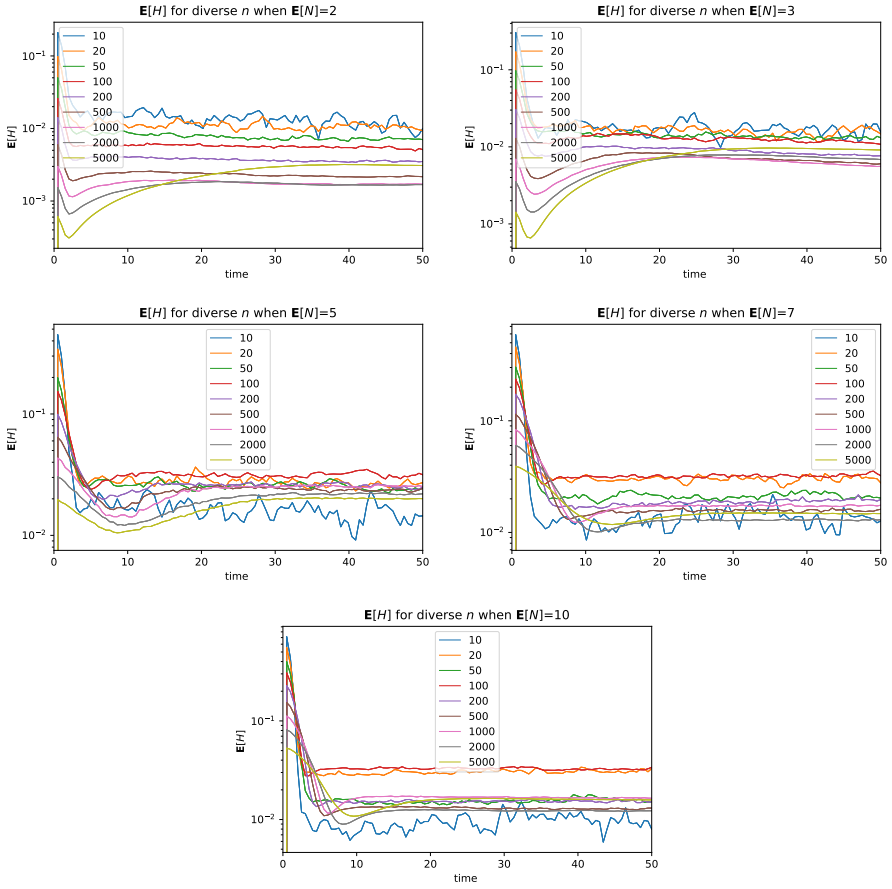

Figure 3: Mean absolute error for diverse network sizes (different colours) for increasing expected neighbourhood sizes (left to right, top to bottom). Denser networks feature a peak in error during the initial stabilisation phase, followed by a quicker convergence due to shorter network diameter. Very small networks have erratic behaviour, larger networks feature a smoother convergence to a lower error.

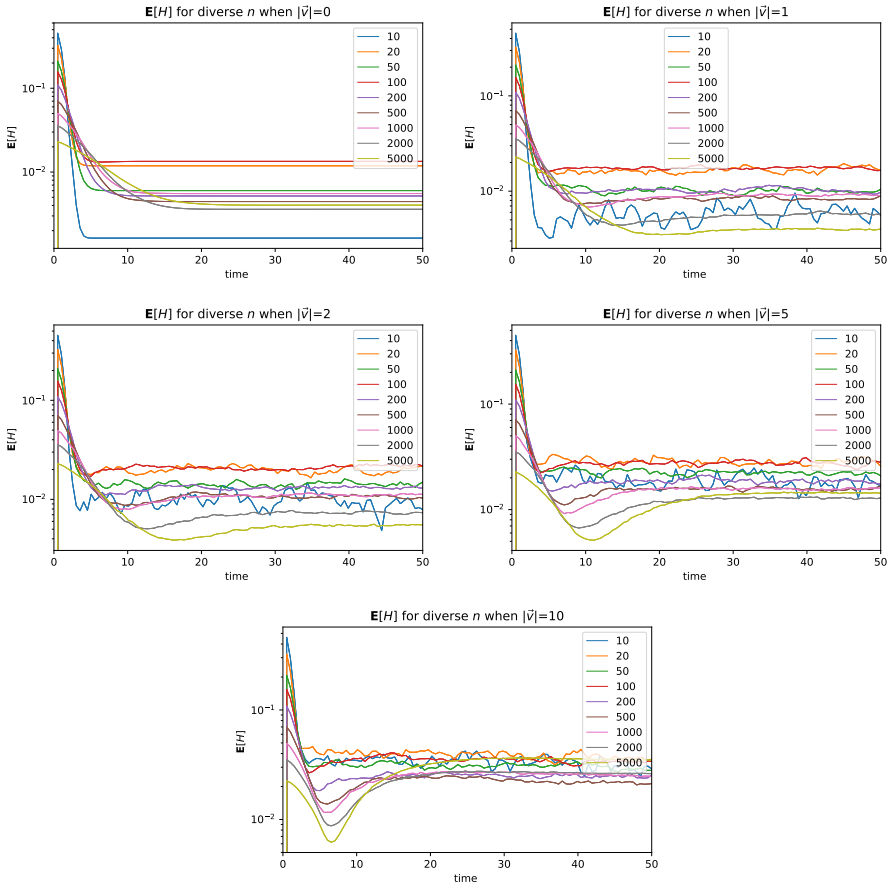

Figure 4: Mean absolute error for diverse network sizes (different colours) for increasing node mobility speed (left to right, top to bottom). Node movement introduces both error and volatility. The larger the network, the more such effect is visible. The algorithm tolerates slow, continuous changes in the network structure rather well, while error ramps up consistently if the network is very instable, possibly due to problem in stabilizing related to the recursive nature of the algorithm.

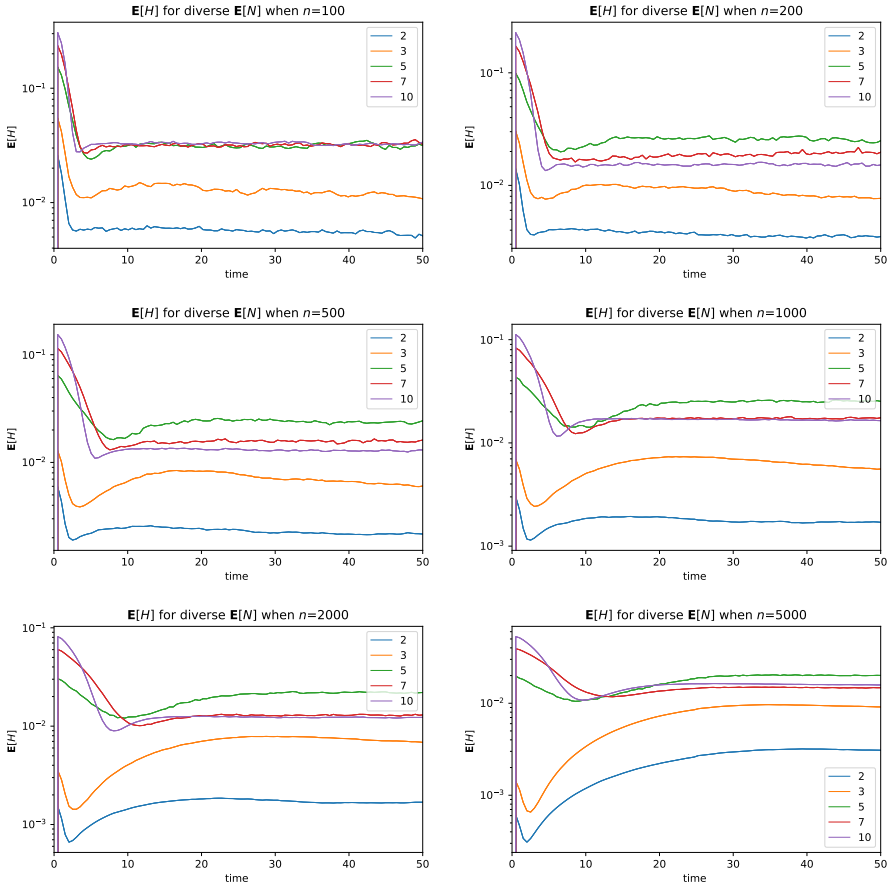

Figure 5: Mean absolute error for diverse expected neighbourhood sizes (different colours) for increasing node count (left to right, top to bottom). Larger network introduce larger error, due to progressive lack of precision in the HyperLogLog estimates. Error peaks for an expected neighbourhood size of five, for which the computation gets performed on a large but sparse network.

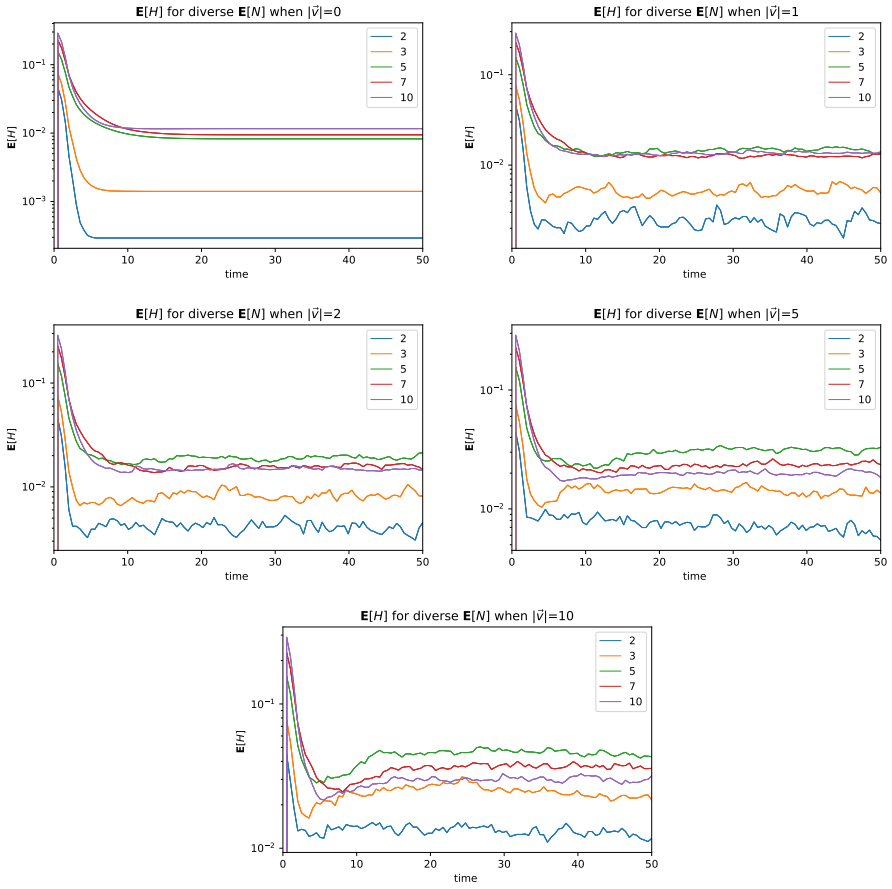

Figure 6: Mean absolute error for diverse expected neighbourhood sizes (different colours) for increasing movement speed (left to right, top to bottom). Node movement introduces both error and volatility, however the network tolerates moderate changes well. Error peaks for an expected neighbourhood size of five, for which the computation gets performed on a large but sparse network.

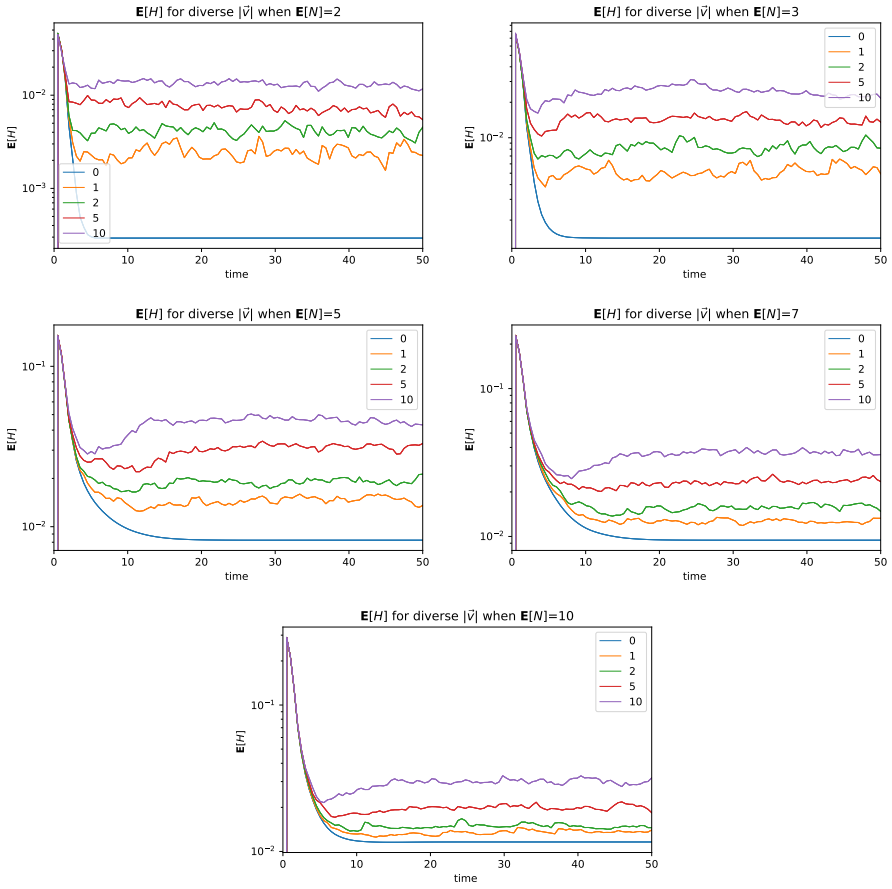

Figure 7: Mean absolute error for diverse node speed (different colours) for increasing expected neighbourhood sizes (left to right, top to bottom). Node movement introduces both error and volatility, however the network tolerates it well. Dense networks converge more quickly regardless of node movement speed, but they have a greater error peak in the initial transient. Higher node mobility impacts absolute error and transient dynamics, but not stabilization time. An initial error peak is measured in all conditions.

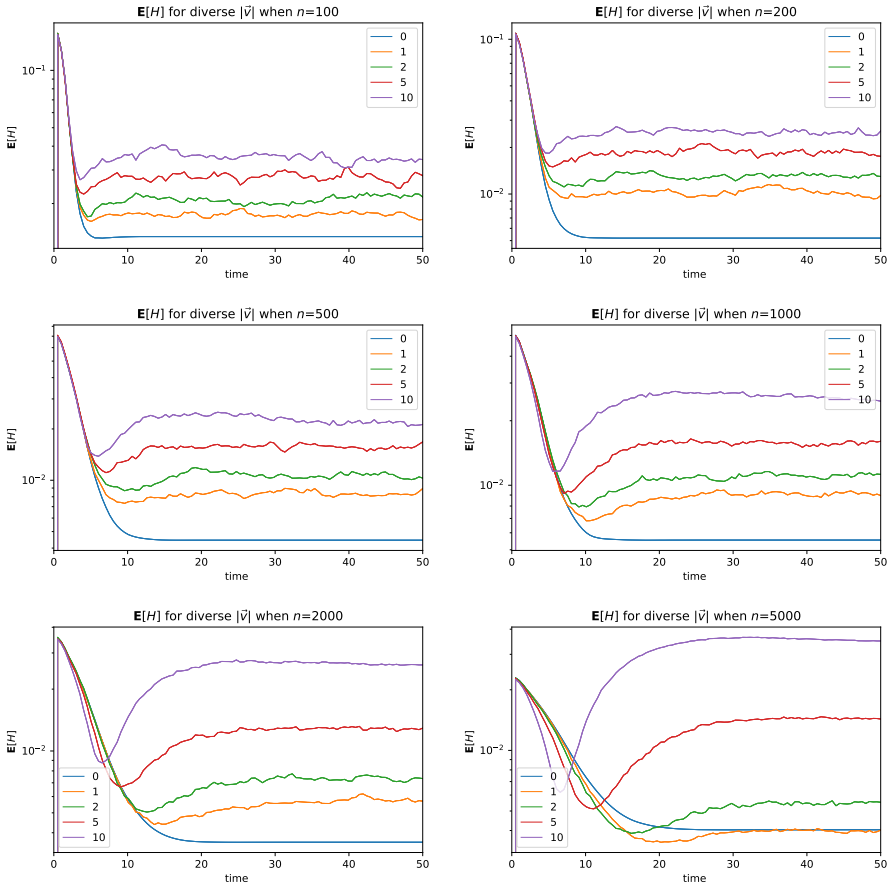

Figure 8: Mean absolute error for diverse node speed (different colours) for increasing network size (left to right, top to bottom). Data shows an increase in error related to increased node movement speed. The first phase of the stabilization transient is dominated by initialization of the computation, which peaks and decreases. The second phase is instead dominated by the error introduced by node movement, which starts low and progressively increases to a plateau.

are interested in two measures: the first is the total count of colonies that get dynamically formed $\mathcal{C}$; and the second, $\overline{IntradDist}$ [27], is the mean across all colonies of the mean distance between every node of a colony and the colony leader. For both measures, lower values indicate better performance: lower $\mathcal{C}$ values suggest a selection of leaders that better covers the overall network extension, while $\overline{IntradDist}$ can be seen as a proxy metric for intra-colony communication latency.

We evaluate the behaviour of different metrics computed with our distributed HyperANF implementation in the following conditions:

- different graph sizes, with the node count n ranging in $\{25, 50, 100, 200, 400, 800\}$;
- different graph topologies: Barabasi-Albert [46], lobster [47], and random euclidean [48];
- different desired cluster radius, ranging (in hops) from 1 to 10.

Graph generation has been realised via GraphStream [49]; in order to run our software stack on the generated topologies, we had to provide integration between Alchemist and GraphStream. Such integration is a further contribution of this work, it has been opensourced and integrated in the simulator since development version `9.3.0-dev1bn+1fc9a8ebd`, and will be part of the next major stable release of Alchemist.

Colony count results are depicted in Figure 9 for the Barabasi-Albert topology, in Figure 10 for the lobster topology, and in Figure 11 for the random euclidean topology. Different topologies have a very relevant effect on performance. The centrality measure that seem to perform better across the board is harmonic, which gets bested in few configurations, and usually of a short margin. Degree centrality performs well on highly connected topologies, such as Barabasi-Albert and random euclidean. It struggles however on lobster topologies, unless the selected radius matches exactly the maximum distance allowed for "non hub" nodes.

Mean intra-cluster distance results are depicted in Figure 12 for the Barabasi-

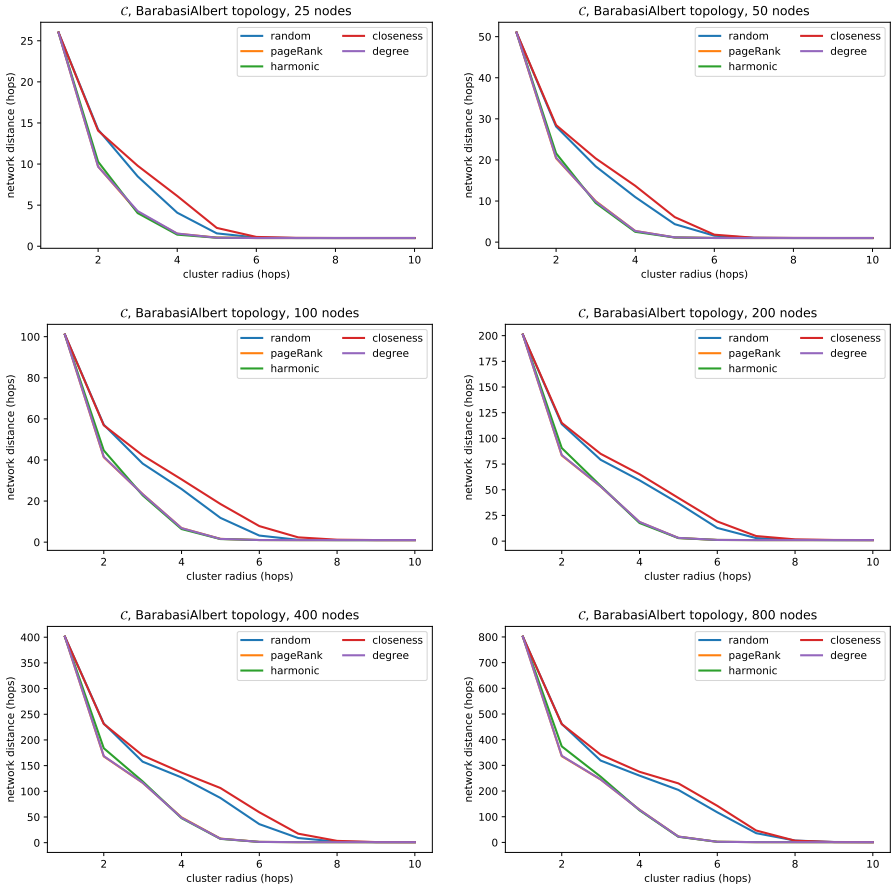

Figure 9: Mean colony count for diverse desired cluster radii on a Barabasi-Albert topology. Degree and harmonic centrality are the best performer, as they both favor the most connected nodes. Closeness centrality is the worst performer, doing slightly worse than a random selection.

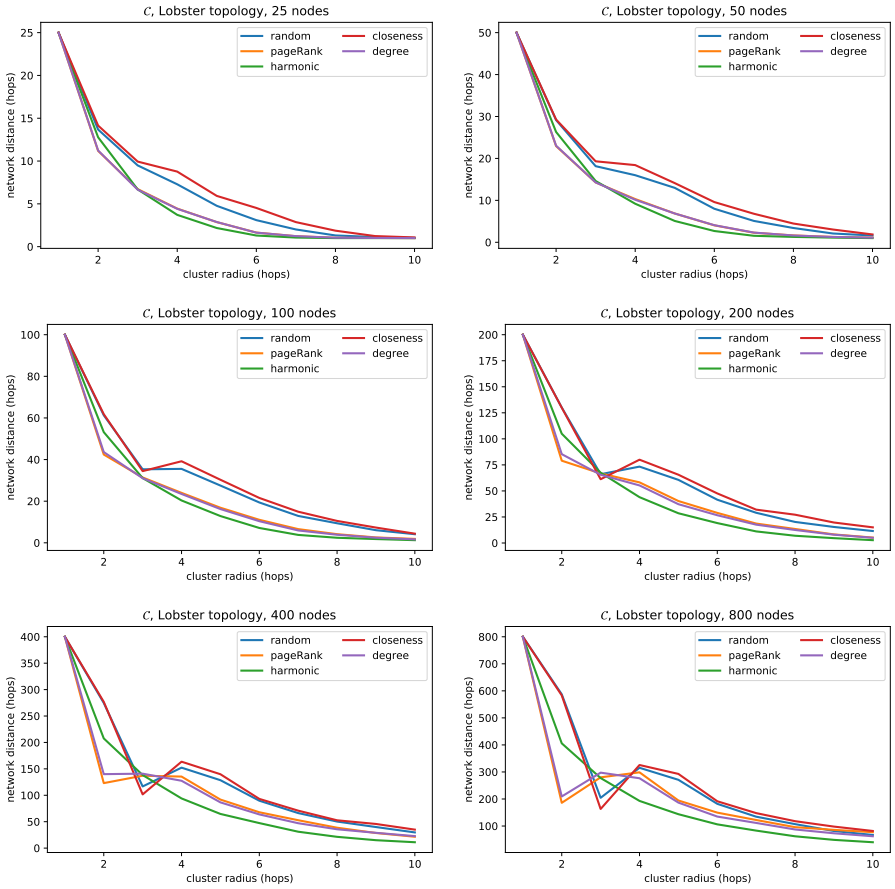

Figure 10: Mean colony count for diverse desired cluster radii on a lobster topology. Harmonic centrality achieves the best performance when large colonies are desired. On large networks, page rank and degree centrality achieve better performance when 2 hops colonies are desired, while closeness and random both outperform harmonic centrality when the desired radius is 3 hops. This phenomenon is strictly related to the specific instance of the lobster graph, featuring at most 2 hops from "hub" nodes, and a maximum degree of 10.

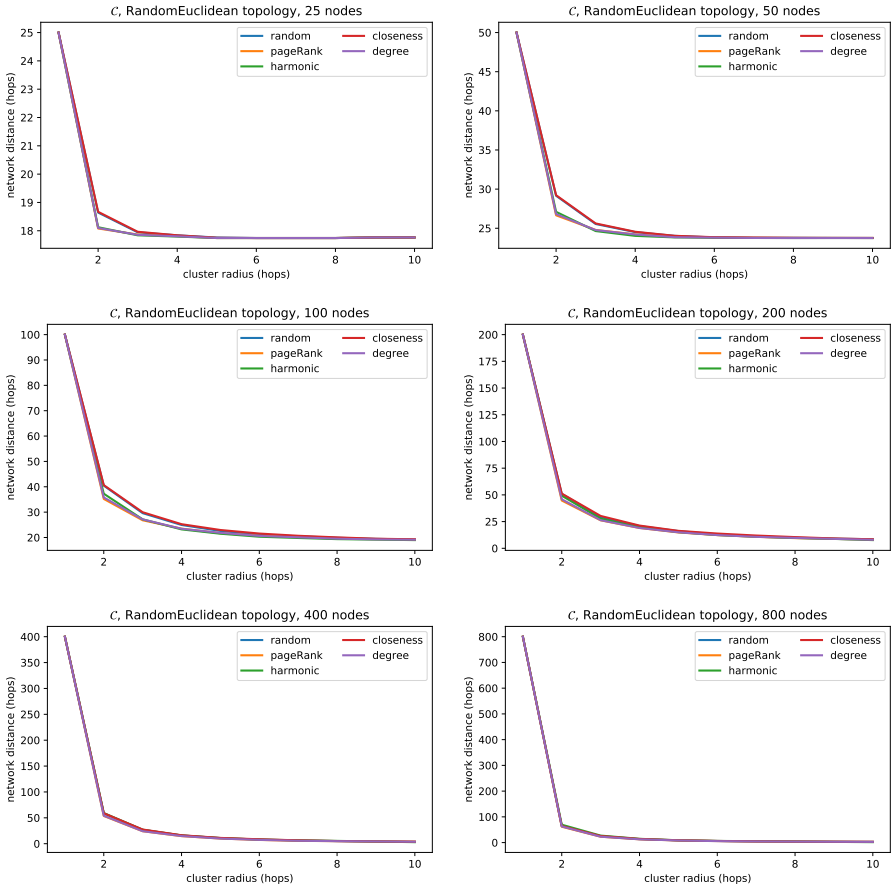

Figure 11: Mean colony count for diverse desired cluster radii on a random euclidean topology. This kind of graphs are very densely connected, and as such performance of all the algorithms (including random selection) are pretty similar, especially on large networks. On smaller network, and for small colonies, pagerank, harmonic, and degree centralities perform better than random and closeness centrality measures.

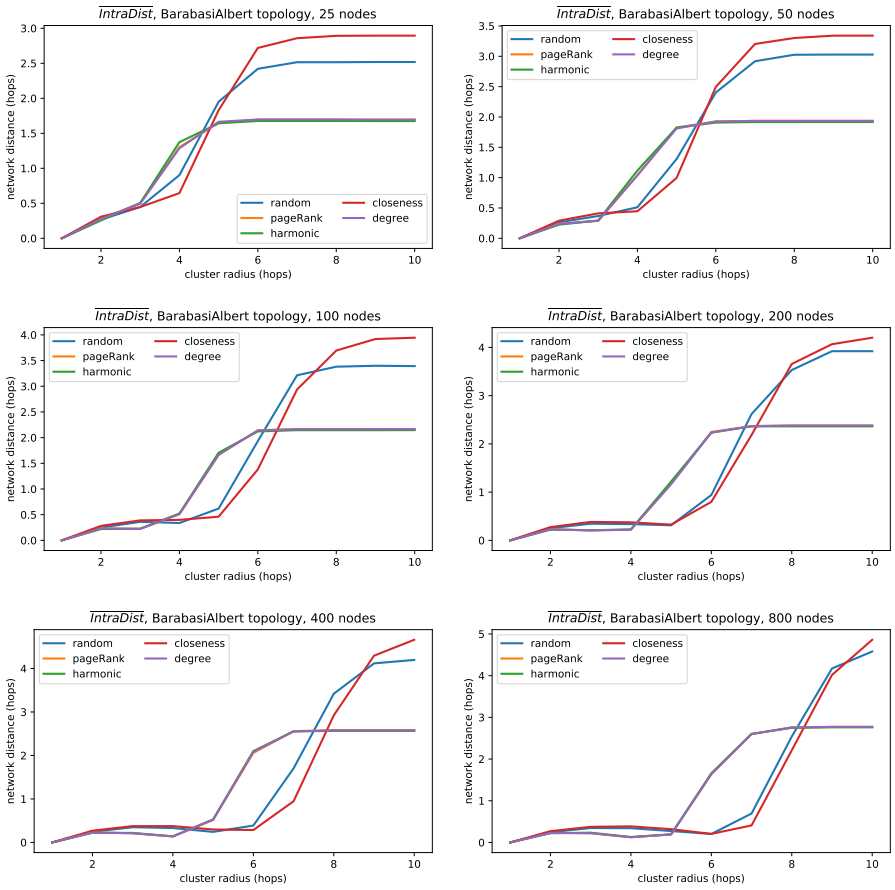

Figure 12: Mean intra-cluster distance across all colonies on a Barabasi-Albert topology. Page rank, harmonic, and degree centrality perform similarly, random and closeness are generally the worst performers. Interestingly, however, for mid-range radii, they seem to outperform the other centrality measures. This effect is due to these centrality measures being source of a larger number of clusters (see Figure 9), whose size (and hence, intra-cluster distance) is smaller.

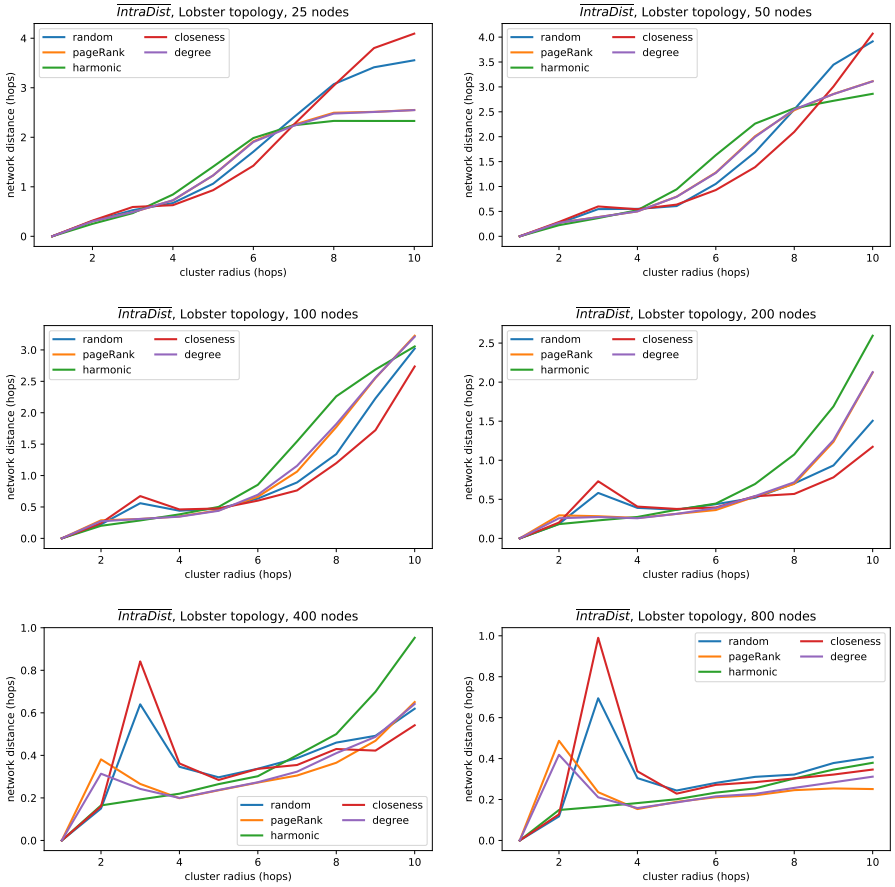

Figure 13: Mean intra-cluster distance across all colonies on a lobster topology. Page rank, harmonic, and degree centrality perform similarly, random and closeness are generally the worst performers. All centrality measure seem to follow a sigmoid shaped curve. As in Figure 9, those conditions in which the centrality measures originate less clusters are linked with a higher intra-cluster distance. However, it is interesting to see how degree outperforms pageRank for colony radius equal to 2, indicating that the former is better at selecting a good "center" for the colony.

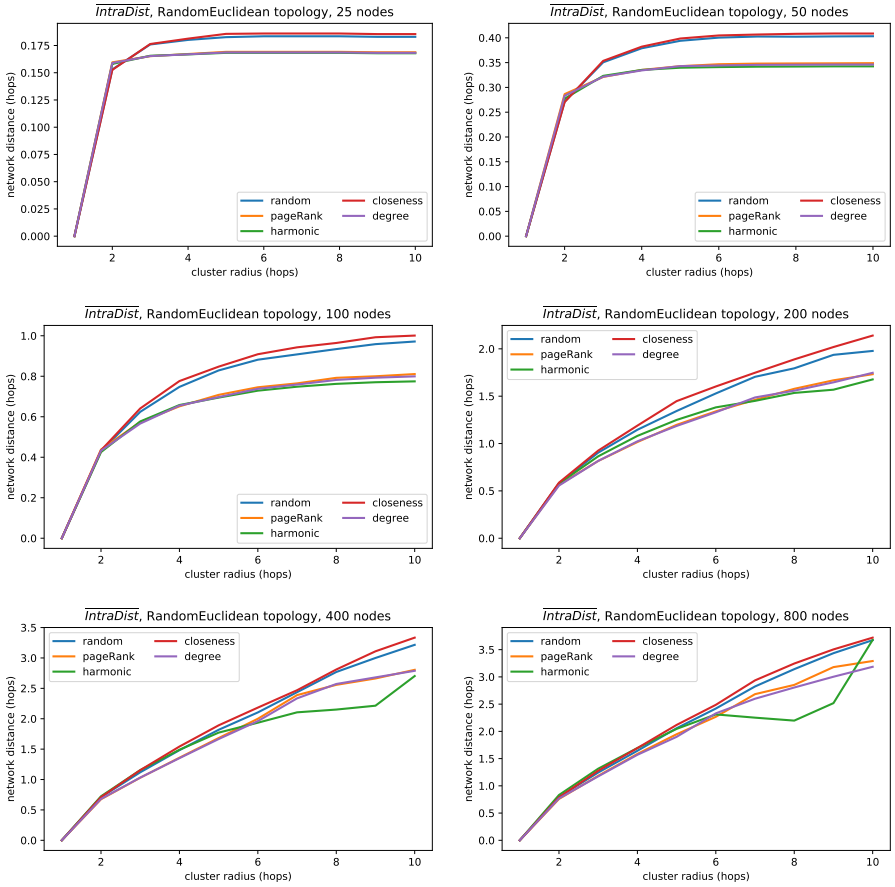

Figure 14: Mean intra-cluster distance across all colonies on a random euclidean topology. In this case, as pictured in Figure 11, the colony count is simular for all centrality measures, so different performances in this context point to a better selection of the leader node. Harmonic centrality is the best performer, closely followed by page rank and degree centrality, which perform very similarly. Closeness centrality performs worse than a random selection.

Albert topology, in Figure 13 for the lobster topology, and in Figure 14 for the random euclidean topology. Results are to be read along with the colony count: in fact, on graphs of the same size, more the colonies are usually associated with smaller colonies and hence shorter intra distances. Of course, however, a smarter selection of the leader node may provide a lower intra distance with a similar colony count. Data shows that harmonic centrality is one of the best performers, closely followed by page rank and degree centrality.

4.3. Application: Self-organising Coordination Regions

In Section 4.2, we discussed how the leader selection (and subsequent fog colony formation) may in principle benefit of a "good" selection of leaders. In this section, we provide empiric evidence that such selection is indeed key for the performance some high level distributed patterns. For instance, in the self-organising coordination regions (SCR) [37] pattern, devices are partitioned, a leader is elected for each partition, and upstream and downstream communication from and to the leader is established. One possible way to select a leader in a partition is by relying on its centrality, with the goal of shortening the latency of information aggregation and diffusion. A typical aggregate programming implementation of SCR would involve the use of single-path converge cast to build upstream (from leaf nodes to the leader) information flow. This algorithm is sensible to the leader position in the network: the longer is the communication delay from the leader to the farthest node in the partition, the higher will be the overall delay (hence, the error) of the whole algorithm.

In this section, we show how a different selection method for the leader may impact the responsiveness of the system by simulating a network partition of $n = 1000$ devices displaced in a regular grid of shape is $d \times \frac{n}{d}$ communicating with their Moore neighborhood (hence, d is both the width and the diameter of the network). We let d range in the set of divisors of n whose value is greater than $\sqrt{n}$, namely, we test grids ranging from 40×25 to 1000×1 devices.

Devices compute a graph statistics on the network and use it to elect a single leader. The leader aggregates the minimum time in the network, in such a way

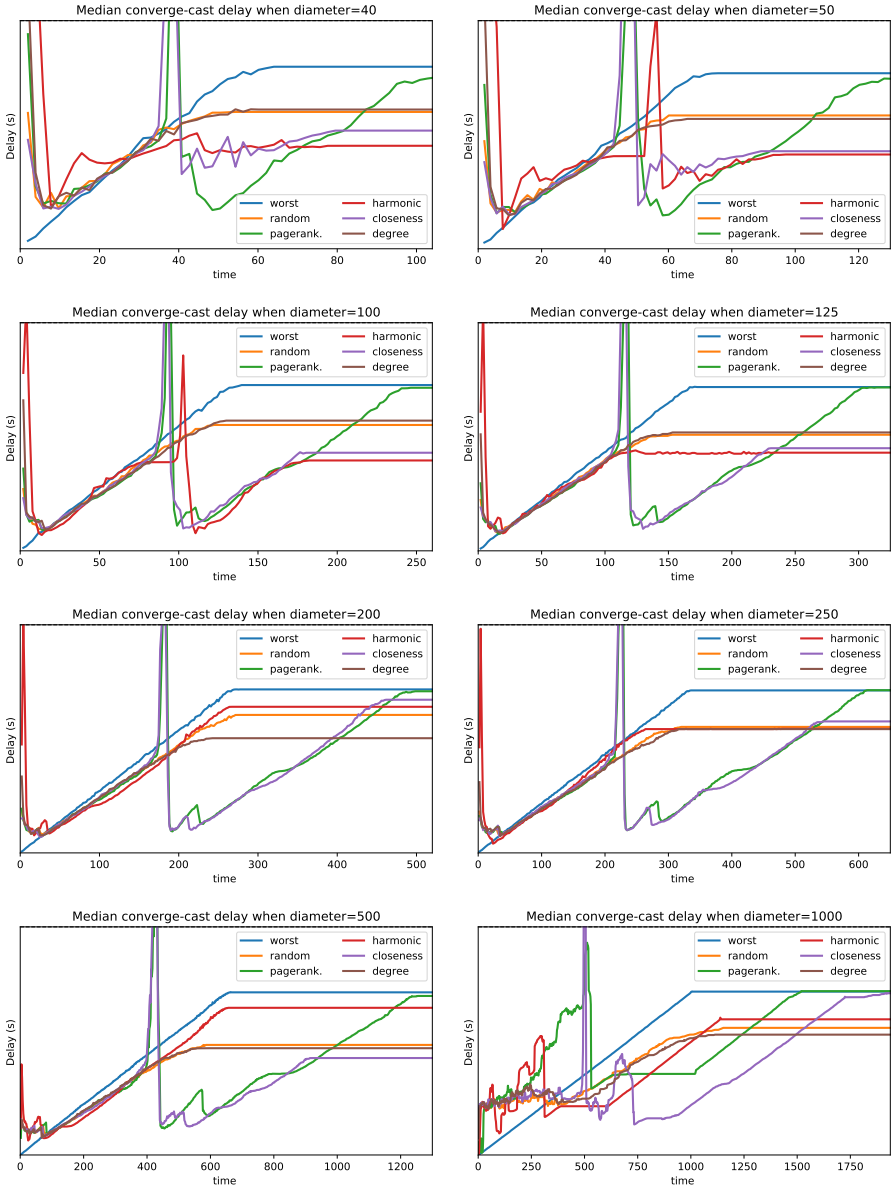

Figure 15: Median delay for converge-cast with different algorithms for leader election for different network shapes, ranging from a 40x25 (top left) to a 1000x1 grid (bottom right). Pagerank performs badly across the board, with a performance comparable to the worst case. Degree centrality's results show a behaviour only slightly improving over a random selection. Harmonic centrality outperfoms every other algorithm in grids up to 125×8, followed closely by closeness centrality. In the most extreme quasi-linear scenarios, degree centrality attains the best results.

that it collects the oldest valid message in the partition. The difference between the collected value and the time at the leader is the information delay. We compare several centrality statistics in fig. 15 (PageRank [21], degree, closeness and harmonic centrality, see section 2.3), using a random selection of the leader and the worst case (a leader on a corner of the grid) as baselines. Notice that random selection is the most common way a leader is selected (especially when devices are homogeneous). Data shows that for bidimensional network shapes closeness centrality and harmonic centrality outperform a random selection of the leader consistently (up to 125×8). Such advantage progressively lessen with more extreme shapes, with degree centrality attaining the best results for quasi-linear 1D networks; which is however an unlikely configuration in edge scenarios.

5. Related Work

This papers concerns the implementation of centrality measure algorithms, and applys to graphs induced by the proximity relation of devices in distributed IoT contexts. To the best of our knowledge, centrality measures have been so far mostly considered for social networks, and their implementation assumes the availability of the (static) graph at hand at a single machine [1, 21, 24], such that the algorithms computes its output using techniques that exploit parallelism to address large-scale. The work in this paper is different from at least two points of view:

- the input graph is not available in a single machine, but is actually distributed across space (each node of the network knows the neighbours it can communicate to): accordingly, computation of a centrality measure is carried on in a fully distributed way, propagating the relevant information until reaching the desired result in each node of the network;
- the input graph can also vary over time (because of device faults, network partitions, and mobility): accordingly, computation has to be seen as an

ongoing task, in which changes to the input are continuosly processed and eventually reflect into the output.

Systems performing this kind of input/output distributed computation are often referred to as self-stabilising systems [3, 50, 51, 52]. There, the goal is to make transient changes eventually irrelevant, such that as soon as a network becomes static, the result converges to the correct expected value.

Several approaches have been introduced to conveniently analyse and design distributed computations spanning over physical space. They fall under the umbrella of so-called *spatial computing languages*—a survey on which can be found in [53], along with a more recent historical perspective in [54]. In principle, most of them could be used to implement the centrality measures as proposed in this paper, though our choice of field-calculus and Protelis language provides a key combination that effectively allows to design and test several centrality measures, making them readily available in practical systems—e.g, for the applications contexts described in [54].

For instance, a number of approaches in spatial computing are aimed at providing programming models that abstract from the existence of individual devices, such as Hood [55], Abstract Regions [56], self-healing geometries [57], or universal patterns [58]: so, they are not appropriate for centrality measures, which are tightly connected to the problem of counting neighbours and estimating distances.

Another set of approaches are more data oriented, and concern summarisation of information extracted from certain regions, to be streamed to others, such as TinyDB [59], Cougar [60], TinyLime [61], and Regiment [62]): again, this context does not fit centrality measures, which has the goal of creating a result in every device of the network, achieved by local propagation and aggregation of information.

Centrality measures require an expressive programming model, specifically targeted at the idea of considering as result of computation a true distributed data structure, namely, one assigning a value to each device of the network.

This is precisely the goal of aggregate computing approaches [4], and especially of field-based computing models, such as those based on the field calculus [5]. While in principle several instantiations of this approach exist, such as Scafi [32], Proto [63], and the fixpoint framework in [7], the programming language Protelis [15] along with the Alchemist simulator [28] provided an expressive and practical framework for quickly designing and testing centrality measure algorithms.

6. Conclusions and Further Work

In this paper, we presented a natural translation of various centrality measures in field calculus, both suggesting that field calculus is well-suited for expressing massively parallel computations over graphs, and producing a new self-stabilising building block which can be applied to solve problems such as improved leader election or network vulnerabilities detection. We then evaluated the translations in two simulation scenarios: the first measuring the error of harmonic centrality estimations in a mutable scenario; the second comparing the effectiveness of various centrality measures in guiding the selection of a central coordinator.

Our results suggest a possible trade-off between computation cost and ranking effectiveness: *PageRank* is more easy to compute, while harmonic centrality is more effective in selecting central vertices. Thus, a centrality measure which is simultaneously effective on IoT graphs and of low computation cost still has to be developed. In future work, we plan to search for centrality measures improving the trade-off, and investigate further practical applications of centrality measures in edge computing case studies. Furthermore, we plan check whether the field calculus implementation of centrality measures may be fruitfully applicable also to HPC settings. Finally, self-stabilisation of PageRank has only been validated in simulation: in future work we plan to search for a formal proof of this fact.

References

[1] P. Boldi, M. Rosa, S. Vigna, Hyperanf: approximating the neighbourhood function of very large graphs on a budget, in: Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011, 2011, pp. 625–634. doi:10.1145/1963405.1963493.

[2] C. R. Palmer, P. B. Gibbons, C. Faloutsos, ANF: a fast and scalable tool for data mining in massive graphs, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada, 2002, pp. 81–90. doi:10.1145/775047.775059.

[3] M. Viroli, G. Audrito, J. Beal, F. Damiani, D. Pianini, Engineering resilient collective adaptive systems by self-stabilisation, ACM Transactions on Modelling and Computer Simulation (TOMACS) 28 (2) (2018) 16:1–16:28. doi:10.1145/3177774.

[4] J. Beal, D. Pianini, M. Viroli, Aggregate programming for the Internet of Things, IEEE Computer 48 (9).

[5] G. Audrito, M. Viroli, F. Damiani, D. Pianini, J. Beal, A higher-order calculus of computational fields, ACM Transactions on Computational Logic (TOCL) 20 (1) (2019) 5:1–5:55. doi:10.1145/3285956.

[6] M. Mamei, F. Zambonelli, Programming pervasive and mobile computing applications with the tota middleware, in: Pervasive Computing and Communications, 2004, IEEE, 2004, pp. 263– 273. doi:10.1109/PERCOM.2004.1276864.

[7] A. Lluch-Lafuente, M. Loreti, U. Montanari, Asynchronous distributed execution of fixpoint-based computational fields, Logical Methods in Computer Science 13 (1). doi:10.23638/LMCS-13(1:13)2017.

[8] J. Bachrach, J. Beal, J. McLurkin, Composable continuous space programs for robotic swarms, Neural Computing and Applications 19 (6) (2010) 825–847.

[9] J. Coutaz, J. L. Crowley, S. Dobson, D. Garlan, Context is key, Commun. ACM 48 (3) (2005) 49–53. doi:10.1145/1047671.1047703.

[10] N. Bicocchi, M. Mamei, F. Zambonelli, Self-organizing virtual macro sensors, TAAS 7 (1) (2012) 2:1–2:28.

[11] J. Beal, K. Usbeck, J. Loyall, M. Rowe, J. Metzler, Adaptive opportunistic airborne sensor sharing, ACM Trans. Auton. Adapt. Syst. 13 (1) (2018) 6:1–6:29.

[12] G. Audrito, F. Damiani, M. Viroli, Aggregate graph statistics, in: D. Pianini, G. Salvaneschi (Eds.), Proceedings First Workshop on Architectures, Languages and Paradigms for IoT, ALP4IoT@iFM 2017, Turin, Italy, September 18, 2017, Vol. 264 of EPTCS, 2017, pp. 18–22. doi:10.4204/EPTCS.264.2.

[13] G. Audrito, J. Beal, F. Damiani, M. Viroli, Space-time universality of field calculus, in: Coordination Models and Languages, Vol. 10852 of Lecture Notes in Computer Science, Springer, 2018, pp. 1–20. doi:10.1007/978-3-319-92408-3_1.

[14] A. Igarashi, B. C. Pierce, P. Wadler, Featherweight Java: A minimal core calculus for Java and GJ, ACM Transactions on Programming Languages and Systems 23 (3).

[15] D. Pianini, M. Viroli, J. Beal, Protelis: Practical aggregate programming, in: ACM Symposium on Applied Computing 2015, 2015, pp. 1846–1853.

[16] M. Eysholdt, H. Behrens, Xtext: Implement your language faster than the quick and dirty way, in: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and

Applications Companion, OOPSLA '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 307–309. doi:10.1145/1869542.1869625.
URL https://doi.org/10.1145/1869542.1869625

[17] L. Bettini, Implementing domain-specific languages with Xtext and Xtend, Packt Publishing Ltd, 2016.

[18] R. Casadei, G. Fortino, D. Pianini, W. Russo, C. Savaglio, M. Viroli, Modelling and simulation of opportunistic iot services with aggregate computing, Future Gener. Comput. Syst. 91 (2019) 252–262. doi:10.1016/j.future.2018.09.005.
URL https://doi.org/10.1016/j.future.2018.09.005

[19] A. Paulos, S. Dasgupta, J. Beal, Y. Mo, K. D. Hoang, L. J. Bryan, P. P. Pal, R. E. Schantz, J. Schewe, R. K. Sitaraman, A. Wald, C. Wayllace, W. Yeoh, A framework for self-adaptive dispersal of computing services, in: IEEE 4th International Workshops on Foundations and Applications of Self* Systems, FAS*W@SASO/ICCAC 2019, Umea, Sweden, June 16-20, 2019, IEEE, 2019, pp. 98–103. doi:10.1109/FAS-W.2019.00036.
URL https://doi.org/10.1109/FAS-W.2019.00036

[20] J. Beal, K. Usbeck, J. P. Loyall, M. Rowe, J. M. Metzler, Adaptive opportunistic airborne sensor sharing, ACM Trans. Auton. Adapt. Syst. 13 (1) (2018) 6:1–6:29. doi:10.1145/3179994.
URL https://doi.org/10.1145/3179994

[21] L. Page, Method for node ranking in a linked database, uS Patent 6,285,999 (Sep. 4 2001).

[22] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Comput. Networks 30 (1-7) (1998) 107–117. doi:10.1016/S0169-7552(98)00110-X.

[23] P. Flajolet, É. Fusy, O. Gandouet, F. Meunier, Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm, in: Analysis of Algorithms 2007 (AofA07), 2007, pp. 127–146.

[24] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, S. Vigna, Four degrees of separation, in: Web Science 2012, WebSci '12, Evanston, IL, USA - June 22 - 24, 2012, 2012, pp. 33–42. doi:10.1145/2380718.2380723.

[25] M. Viroli, R. Casadei, D. Pianini, On execution platforms for large-scale aggregate computing, in: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct, UbiComp '16, ACM, New York, NY, USA, 2016, pp. 1321–1326. doi:10.1145/2968219.2979129.

[26] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113. doi:10.1145/1327452.1327492.

[27] C. Guerrero, I. Lera, C. Juiz, On the influence of fog colonies partitioning in fog application makespan, in: M. Younas, J. P. Disso (Eds.), 6th IEEE International Conference on Future Internet of Things and Cloud, FiCloud 2018, Barcelona, Spain, August 6-8, 2018, IEEE Computer Society, 2018, pp. 377–384. doi:10.1109/FiCloud.2018.00061.
URL https://doi.org/10.1109/FiCloud.2018.00061

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

[29] M. A. Gibson, J. Bruck, Efficient exact stochastic simulation of chemical systems with many species and many channels, The Journal of Physical Chemistry A 104 (9) (2000) 1876–1889. doi:10.1021/jp993732q.
URL https://doi.org/10.1021/jp993732q

[30] R. Casadei, M. Viroli, G. Audrito, D. Pianini, F. Damiani, Engineering collective intelligence at the edge with aggregate processes, Engineering Applications of Artificial Intelligence 97 (104081). doi:10.1016/j.engappai.2020.104081.

[31] S. Montagna, D. Pianini, M. Viroli, A model for drosophila melanogaster development from a single cell to stripe pattern formation, in: S. Ossowski, P. Lecca (Eds.), Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, March 26-30, 2012, ACM, 2012, pp. 1406–1412. doi:10.1145/2245276.2231999.
URL https://doi.org/10.1145/2245276.2231999

[32] R. Casadei, M. Viroli, Towards aggregate programming in Scala, in: First Workshop on Programming Models and Languages for Distributed Computing, PMLDC '16, ACM, New York, NY, USA, 2016, pp. 5:1–5:7. doi:10.1145/2957319.2957372.

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

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

[35] V. Zaburdaev, S. Denisov, J. Klafter, Lévy walks, Reviews of Modern Physics 87 (2) (2015) 483–530. doi:10.1103/revmodphys.87.483.

[36] S. Mertens, C. Moore, Continuum percolation thresholds in two dimensions, Physical Review E 86 (6).

[37] D. Pianini, R. Casadei, M. Viroli, A. Natali, Partitioned integration and coordination via the self-organising coordination regions pattern, Future Generation Computer Systems 114 (2021) 44 – 68. doi:https://doi.org/10.1016/j.future.2020.07.032.
URL http://www.sciencedirect.com/science/article/pii/S0167739X20304775

[38] J. Wang, Y. Gao, K. Wang, A. K. Sangaiah, S. Lim, An affinity propagation-based self-adaptive clustering method for wireless sensor networks, Sensors 19 (11) (2019) 2579. doi:10.3390/s19112579.
URL https://doi.org/10.3390/s19112579

[39] E. Sakhaee, K. Leibnitz, N. Wakamiya, M. Murata, Bio-inspired layered clustering scheme for self-adaptive control in wireless sensor networks, in: 2009 2nd International Symposium on Applied Sciences in Biomedical and Communication Technologies, 2009, pp. 1–6.

[40] S. Raghuwanshi, A. Mishra, A self-adaptive clustering based algorithm for increased energy-efficiency and scalability in wireless sensor networks, in: 2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484), Vol. 5, 2003, pp. 2921–2925 Vol.5.

[41] P. Zahadat, Self-adaptation and self-healing behaviors via a dynamic distribution process, in: IEEE 4th International Workshops on Foundations and Applications of Self* Systems, FAS*W@SASO/ICCAC 2019, Umea, Sweden, June 16-20, 2019, IEEE, 2019, pp. 261–262. doi:10.1109/FAS-W.2019.00072.
URL https://doi.org/10.1109/FAS-W.2019.00072

[42] R. L. Stewart, R. A. Russell, A distributed feedback mechanism to regulate wall construction by a robotic swarm, Adapt. Behav. 14 (1) (2006) 21–51. doi:10.1177/105971230601400104.
URL https://doi.org/10.1177/105971230601400104

[43] M. Abderrahim, M. Ouzzif, K. Guillouard, J. François, A. Lebre, A holistic monitoring service for fog/edge infrastructures: A foresight study, in: M. Younas, M. Aleksy, J. Bentahar (Eds.), 5th IEEE International Conference on Future Internet of Things and Cloud, FiCloud 2017, Prague, Czech Republic, August 21-23, 2017, IEEE Computer Society, 2017, pp. 337–344. doi:10.1109/FiCloud.2017.30.
URL https://doi.org/10.1109/FiCloud.2017.30

[44] A. Brogi, S. Forti, M. Gaglianese, Measuring the fog, gently, in: S. Yangui, I. B. Rodriguez, K. Drira, Z. Tari (Eds.), Service-Oriented Computing - 17th International Conference, ICSOC 2019, Toulouse, France, October 28-31, 2019, Proceedings, Vol. 11895 of Lecture Notes in Computer Science, Springer, 2019, pp. 523–538. doi:10.1007/978-3-030-33702-5_40.
URL https://doi.org/10.1007/978-3-030-33702-5_40

[45] O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, Towards qos-aware fog service placement, in: 1st IEEE International Conference on Fog and Edge Computing, ICFEC 2017, Madrid, Spain, May 14-15, 2017, IEEE Computer Society, 2017, pp. 89–96. doi:10.1109/ICFEC.2017.12.
URL https://doi.org/10.1109/ICFEC.2017.12

[46] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512. doi:10.1126/science.286.5439.509.
URL https://doi.org/10.1126/science.286.5439.509

[47] X. Zhou, B. Yao, X. Chen, Every lobster is odd-elegant, Information Processing Letters 113 (1-2) (2013) 30–33. doi:10.1016/j.ipl.2012.09.008.
URL https://doi.org/10.1016/j.ipl.2012.09.008

[48] M. Penrose, et al., Random geometric graphs, Vol. 5, Oxford university press, 2003.

[49] Y. Pigné, A. Dutot, F. Guinand, D. Olivier, Graphstream: A tool for bridging the gap between complex systems and dynamic graphs, CoRR abs/0803.2093. arXiv:0803.2093.
URL http://arxiv.org/abs/0803.2093

[50] K. Altisen, P. Corbineau, S. Devismes, A framework for certified self-stabilization, in: Formal Techniques for Distributed Objects, Components, and Systems, Springer International Publishing, Cham, 2016, pp. 36–51.

[51] F. Faghih, B. Bonakdarpour, S. Tixeuil, S. Kulkarni, Specification-based synthesis of distributed self-stabilizing protocols, in: Formal Techniques

for Distributed Objects, Components, and Systems, Springer International Publishing, Cham, 2016, pp. 124–141.

[52] J. Beal, M. Viroli, Building blocks for aggregate programming of self-organising applications, in: 2nd FoCAS Workshop on Fundamentals of Collective Systems, IEEE CS, to appear, 2014, pp. 1–6.

[53] J. Beal, S. Dulman, K. Usbeck, M. Viroli, N. Correll, Organizing the aggregate: Languages for spatial computing, in: Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, IGI Global, 2013, Ch. 16, pp. 436–501. doi:10.4018/978-1-4666-2092-6.ch016.

[54] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, D. Pianini, From distributed coordination to field calculus and aggregate computing, J. Log. Algebraic Methods Program. 109. doi:10.1016/j.jlamp.2019.100486.
URL https://doi.org/10.1016/j.jlamp.2019.100486

[55] K. Whitehouse, C. Sharp, E. Brewer, D. Culler, Hood: a neighborhood abstraction for sensor networks, in: Proceedings of the 2nd international conference on Mobile systems, applications, and services, ACM Press, 2004.

[56] M. Welsh, G. Mainland, Programming sensor networks using abstract regions., in: NSDI, Vol. 4, 2004, pp. 3–3.
URL https://dl.acm.org/citation.cfm?id=1251178

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

[58] D. Yamins, A theory of local-to-global algorithms for one-dimensional spatial multi-agent systems, Ph.D. thesis, Harvard, Cambridge, MA, USA (2007).

[59] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong, TAG: A Tiny AGgregation Service for Ad-hoc Sensor Networks, SIGOPS Oper. Syst. Rev. 36 (2002) 131–146. doi:10.1145/844128.844142.

[60] Y. Yao, J. Gehrke, The cougar approach to in-network query processing in sensor networks, SIGMOD Record 31 (2002) 9–18. doi:10.1145/601858.601861.

[61] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, G. P. Picco, Mobile data collection in sensor networks: The tinylime middleware, Elsevier Pervasive and Mobile Computing Journal 4 (2005) 446–469.

[62] R. Newton, M. Welsh, Region streams: Functional macroprogramming for sensor networks, in: Workshop on Data Management for Sensor Networks, DMSN '04, ACM, 2004, pp. 78–87.

[63] J. Beal, J. Bachrach, Infrastructure for engineered emergence in sensor/actuator networks, IEEE Intelligent Systems 21 (2006) 10–19.