Computational Models for Networks of Tiny Artifacts: a Survey[☆]

Carme Àlvarez[a], Ioannis Chatzigiannakis[b], Amalia Duch[a], Joaquim Gabarró[a], Othon Michail[b], Maria Serna[a], Paul G. Spirakis[b]

[a] *Universitat Politècnica de Catalunya (UPC). ALBCOM Research group. Dept. de Llenguatges i Sistemes Informàtics, Barcelona, Spain*
[b] *Research Academic Computer Technology Institute (RACTI) & Computer Engineering and Informatics Department (CEID), University of Patras, Patras, Greece*

Abstract

We survey here some recent computational models for networks of tiny atrifacts. In particular, we focus on networks consisting of artifacts with sensing capabilities. We first imagine the artifacts moving passively, that is, being mobile but unable to control their own movement. This leads us to the *population protocol* model of Angluin *et al* [1]. We survey this model and some of its recent enhancements. In particular, we also present the *mediated population protocol* model in which the interaction links are capable of storing states and the *passively mobile machines* model in which the finite state nature of the agents is relaxed and the agents become multitape Turing machines that use restricted space. We next survey the *sensor field* model, a general model capturing some identifying characteristics of many sensor network's settings. A sensor field is composed of a kind of devices that can communicate one to the other and also to the environment through input/output data streams. We, finally, present simulation results between sensor fields and population protocols and analyze the capability of their variants to decide graph properties.

Keywords: population protocols, mediated population protocols, sensor field, graph languages, sensing problems, sensor networks

1. Introduction

The use of networks of tiny artifacts is becoming a key ingredient in the technological development of XXI century societies. An example of those networks are the networks with sensors, where some of the artifacts have the ability of sensing the environment and communicate among themselves. It is quite usual nowadays to hear about sensor network applications. For instance, the use of sensor networks to study the roaming of herds of endangered species or to control flooding threats or to coordinate mobile communication networks or even to monitor vital signs in human patients and perform highway traffic control.

All these applications and many more are possible due to a wide range of artifacts that become smaller, low-cost, available, extended and wireless interconnected to existing networks which are pervasive (they are embedded in the environment and almost form part of it) and ubiquous (they are everywhere, anytime).

Such networks require several capabilities. Namely, sense, communicate, move, evolve, interact, adapt, act, among others. Therefore, the study of such systems involves several and very different areas of computing: hybrid and distributed systems, communication systems and protocols, circuit design, multi-agent systems, ad-hoc networks, algorithmic design, complexity theory or pervasive and ubiquous computing. Consequently, one can assume that this kind of systems are intrinsically complex. In fact, there is no easy way to design a universal sensor network that acts properly in all possible situations.

However, it is important to understand the computational process and behavior of the different types of artifact's networks, which will help in taking the maximum profit of those networks. In the particular case of networks with sensors several proposals (taxonomies and surveys) that elucidate their distinguishing features and their applications have been published ([2, 3, 4, 5, 6, 7, 8]). These proposals state clearly the need of formal models that capture the clue characteristics of networks with sensors that consist of massive amounts of tiny devices with limited resources.

The general sensing setting can be described by two elements: the observers or end users and the phenomenon, the entity of interest to the observers that is monitored and analyzed by a network with sensors. The corresponding information is discretized in two ways: first the environment is sampled on a discrete set of locations (sensor

[☆] This work has been partially supported by the ICT Programme of the European Union under contract number ICT-2008-215270 (FRONTS). The first, third, fourth, and sixth authors were partially supported by SGR2009-2013 (ALBCOM). The first, fourth, and sixth authors were partially supported by MEC TIN-2007-66523 (FORMALISM).

Email addresses: alvarez@lsi.upc.edu (Carme Àlvarez), ichatz@cti.gr (Ioannis Chatzigiannakis), duch@lsi.upc.edu (Amalia Duch), gabarro@lsi.upc.edu (Joaquim Gabarró), michailo@cti.gr (Othon Michail), mjserna@lsi.upc.edu (Maria Serna), spirakis@cti.gr (Paul G. Spirakis)

positions), and second the measures taken by the sensors are digitalized to the corresponding precision. To analyze the correctness and performance of the system we are faced with a double task; on one side there is a computational problem to be solved by a particular network; on the other hand, it is necessary to assess whether a computed solution is a valid observation of the phenomenon. Both tasks will require different analysis tools and we concentrate here on the first one. The distinctive peculiarities of the computational system define new parameters to be evaluated in order to measure the performance or the stability of the system. Metrics are needed to allow us to estimate the suitability of a specific or generic network topology or the possibility of emergent behavior with pre-specified requirements.

The computational system can be perceived and analyzed in two complementary ways. The first one has as goal to show the emergency of some designed behaviour. In this scenario it is usual to assume that the system models some kind of interaction among the participating devices and the final goal is to achieve a configuration with the pre-specified goals. The task has to be carried out by the network based on the subjacent communication model and the exchange of information among the devices.

For the first, we focus on models coming from the area of population protocols [9, 10, 11]. They represent sensor networks, supposing that the corresponding sensing devices are extremely limited mobile agents, unable to control their own movement, that interact only in pairs by means of an interaction graph. These models bear a strong resemblance to models of interacting molecules in theoretical chemistry [12, 13]. According to [14], the population protocol model was inspired in part by work by Diamadi and Fischer [15] on trust propagation in a social network. The *urn automata* of [16] can be seen as a first draft of the model that retained in vestigial form several features of classical automata: instead of interacting with each other, agents could interact only with a finite-state controller, complete with input tape. The motivation given for the population protocol model proposed in [1] was the study of sensor networks in which passive agents were carried along by other entities; the canonical example was sensors attached to a flock of birds. The name of the model was chosen by analogy to *population processes* [17] in probability theory.

The computational system arising from the ad-hoc computation network point of view can be modeled by combining the notion of graph automata [18] together with distributed data streams [19], a combination inspired in similar ideas developed in the context of concurrent programming [20]. Existing models coming from distributed systems [21], hybrid systems and ad-hoc networks [22, 23] capture some of such networks. This approach has been used also in many papers in which the communication network is assumed to be formed by a random geometric graph [24, 25, 26, 27, 28] and follows classic distributed approaches to solve problems on particular topologies [29]. Based on those ideas the *Sensor Field* model was proposed in [30]. A sensor field captures some characteristic differences of sensor networks, it is composed of a kind of actuator devices that can communicate one to the other and also can measure and signal the environment. The initial study has been concentrated in the case in which the devices and the communication links do not appear and disappear during the computation. The model assumes that those devices synchronize at barriers marking rounds, in a way similar to the BSP model [31]. During a computation round, a device access the received messages and data provided by the environment, performs some computation, and finally sends messages to its neighbors and to the environment. Those are the fundamental features of the *Static and Synchronous Sensor Field* (SSSF) model. The SSSF model can be seen as a non-uniform computational model in the sense that it is easy to introduce constraints to all or some of the devices of the sensor field and relate it to classic complexity classes.

In the remaining of this paper we survey recent results on computational models for networks of tiny artifacts. The survey is organized as follows: In Section 2 we highlight the fundamental problems to be solved in such networks. Section 3 is devoted to the population protocol model and its variants. The sensor field model is reviewed in Section 4. In Section 5 we provide the main ideas used to simulate population protocols by a sensor field. Section 6 is devoted to analyze the graph languages that are decidable by an enhancement of population protocols and some subfamilies of sensor fields with constant memory per device. The last section provides some conclusions and open lines of research.

2. Problems of Interest

Before introducing in detail the models and the results we want to highlight first some classes of problems that are relevant in sensor networks application.

Sensor networks usually consist of massive amounts of cheap, bulk-produced sensor nodes (also called *agents*) and the resources available to each node are most of the time severely limited. Such limitations can be surpassed if the system designer has fine control over the interactions between the agents. In this case, even finite-state agents can be regimented into cellular automata [32] with computational power equivalent to linear space Turing machines. As Angluin *et al.* noticed in [9], if the system designer cannot control these interactions, it is not clear what the computational limits are.

Consider a wireless sensor network in which the sensor nodes move according to some mobility pattern over which they have totally no control. This kind of mobility is known as *passive mobility* [9]. For example, imagine that millions of such nodes are thrown into a hurricane in order to cooperatively study its various characteristics. Some metrics of interest could be the average or maximum barometric pressure, the highest temperature,

or some collective data concerning the wind speed near the eye of the hurricane. In this scenario, the movement of the sensor nodes follows some collection of interchanging probability distributions which, as a result of some natural phenomenon, are in general totally unpredictable (any other phenomenon offering an abundance of kinetic energy could be representative). The nodes have to sense their environment according to the underlying query and then interact with the remaining population in order to make some cooperative decision or cooperatively compute a required function on the sensed inputs. An interaction is established when two nodes come sufficiently close to each other so that communication becomes feasible (one within the range of the other). Communication is usually assumed to be *two-way* though various modes of communication have been studied in the relevant literature [33, 14]. Additionally, the nodes are assumed to communicate in ordered pairs (u, v), where u plays the role of the *initiator* and v that of the *responder*. The distinct roles of the two participants constitutes a fundamental symmetry breaking assumption.

The critical constraints of these systems are clearly memory and the inability of protocols to control interactions. These constraints render most classical distributed algorithms useless under this setting. Even electing a leader and exploiting it to execute a protocol that assumes its existence is clearly non-trivial. The reason is that we may easily elect a leader but, under no probabilistic termination estimates, we will never be able to detect termination of the leader election process. Recent developments [34, 11] relax the memory constraint to allow the existence of unique identifiers (commonly abbreviated as *ids* or *uids*) but no previously existing algorithm can perform the id assignment under this totally asynchronous and pairwise communication setting.

Another set of computational problems of interest consider the sensors as a source of data to be treated by the system. Those *sensing problems* have been stated in a generic way in terms of input/output data streams and a relation that output data streams have to satisfy given the input data streams, i.e. the property that defines the problem [30]. Let us first introduce the notion of data stream.

A *data stream* w is a sequence of data items $w = w^1 w^2 \dots w^i \dots$ that can be infinite. For any $i \geq 1$, $w[i]$ denotes the i-th element of w, i.e. $w[i] = w^i$. For any i, j, $1 \leq i \leq j$, $w[i, j]$ denotes the subsequence of w composed by all data items between the i-th and j-th positions, i.e. $w[i, j] = w^i \dots w^j$. For any $n \geq 1$, an n-data stream $\mathbf{w}$ is an n-tuple of data streams, $\mathbf{w} = (w_1, \dots, w_n)$. For any $i \geq 1$, $\mathbf{w}[i]$ denotes the n-tuple composed by all the i-th elements of each data stream, $\mathbf{w}[i] = (w_1[i], \dots, w_n[i])$. For any i, j such that $1 \leq i \leq j$, $\mathbf{w}[i, j]$ denotes the n-tuple composed by the subsequences between the i-th and j-th positions of each data stream, $\mathbf{w}[i, j] = (w_1[i, j], \dots w_n[i, j])$.

Now we can define formally in an abstract way what a sensing problem is.

Sensing Problem Π: Given an n-tuple of data streams $\mathbf{u} = (u_k)_{1 \leq k \leq n}$ for some $n \geq 1$, compute an m-tuple of data streams $\mathbf{v} = (v_k)_{1 \leq k \leq m}$ for some $m \leq n$ such that $R_\Pi(\mathbf{u}[1, t], \mathbf{v}[1, t])$ is satisfied for every $t \geq 1$. R_Π is the relation that output data streams have to satisfy given the input data streams, i.e. the property that defines the problem.

Sensing problems capture many classes of problems that arise in a natural way in a distributed system. Concerning the applications of sensor networks we consider some sensing problem subfamilies. Those families are obtained by posing additional requirements on the part of the input data stream that is required for computing the t-th element of the output data stream.

- Cumulative monitoring

  $\mathbf{v}[t]$ depends on all the data measured by the network up to time t

- Monitoring

  $\mathbf{v}[t]$ depends only on the data measured at time t

- One step measure

  $\mathbf{v}[t]$ depends only on the data measured at a particular time step, usually the first.

When the computational model per node is a finite state machine, the output data stream can contain the internal state. Thus conditions on the internal state can be transfered to conditions on the output data stream.

Let us post some examples of sensing problems of the three types. First we consider a problem in which it is needed to monitor continuously a wide area. This implies "sensing locally" and "informing locally" about a global environmental phenomena.

Average Monitoring: Given n data streams $(u_k)_{1 \leq k \leq n}$ for some $n \geq 1$, compute n data streams $(v_k)_{1 \leq k \leq n}$ such that $v_k[t] = (u_1[t] + \dots + u_k[t])/n$.

The second example we consider is related to "fire detection alarm". In this case it is desired to detect the situation in which there is a high risk of fire. One element to be measured is the level of smoke in the air of such area and if this level is higher than a certain value then the alert has to be activated. A specific device (number 1 for instance) acts as a master and outputs the result.

Alerting: Given n data streams $(u_k)_{1 \leq k \leq n}$ for some $n \geq 1$, and threshold value A, device 1 has to compute a data stream v_1 such that

$$v_1[t] = \begin{cases} 1 & \text{if } \exists k : 1 \leq k \leq n : u_k[t] \geq A \\ \perp & \text{otherwise} \end{cases}$$

This is a continuous monitoring type of problem as the t-th output data stream is computed from all the previous segment of input data streams. The third example uses artificially the input data stream. The final goal is

to analyze a first step property, therefore only the initial measurement is of relevance.

Initially even?: Given n data streams $(u_k)_{1\leq k\leq n}$ for some $n \geq 1$, such that for $u_k[t] \in \{0,1\}$, for any $1 \leq k \leq n$ and any $t > 0$, compute n data streams $(v_k)_{1\leq k\leq n}$ such that $v_k[t] = [(u_1[1] + \cdots + u_n[1]) \mod 2 = 0]$. Usually for this class of problems we speak rather of an input to the network than of an input data stream for the network.

Observe that in the definition of the problems we have stated properties in which, in general, the t-th output data is related up to the t-th input data. Of course when solving the problem we have to allow some delay in the computation of the system. As we will see this will be one of the relevant parameters when solving the problem with time restrictions (as in the sensor field model). In other models we do not care about the time as the interest is in showing that eventually the required output data stream will be attained (as in the population protocol models). Note that sensing problems are defined in terms of input and output data streams and their definition is independent of the topology of the network.

The second family of problems, the graph decision problems, are related to the network topology. In this case we face the problem of deciding whether the actual subjacent topology (interaction or communication) satisfies some required property. This notion has been formalized as *graph languages* which will be formally described in Section 6.

The goal of deciding graph languages is to determine properties of the subjacent unknown, interaction or communication, graph. Observe, that in this case the general definition of the problem must consider also the subjacent (interaction/communication) graph that is used to solve the problem. As an example lets consider the following problem.

2-cycle?: Compute n data streams $(v_k)_{1\leq k\leq n}$ such that, for ant $t > 0$, $v_k[t] = 1$ iff the subjacent graph contains a two cycle.

In this case, the output data streams have to agree on a repeated symbol that depends on whether the network property is or is not verified by the network. Besides solving information problem about the subjacent topology decidability of graph languages can be used to compare the computational power of different computational models for networks formed by constant memory devices.

So, there was a clear need for some new theoretical framework in order to study the computational power of the population protocol model and its enhancements and to devise protocols for practical problems that need to be solved.

3. Population Protocols & Enhancements

Let us start by defining formally a *population protocol* (PP) [1, 9].

Population Protocol $\mathcal{A}$ (PP $\mathcal{A}$): Formally, we define a population protocol $\mathcal{A}$ by a 6-tuple $\mathcal{A} = (X, Y, Q, I, O, \delta)$, where X, Y, and Q are all finite sets and

1. X is the *input alphabet*,
2. Y is the *output alphabet*,
3. Q is the set of *states*,
4. $I : X \to Q$ is the *input function*,
5. $O : Q \to Y$ is the *output function*, and
6. $\delta : Q \times Q \to Q \times Q$ is the *transition function*.

If $\delta(a, b) = (a', b')$ then $(a, b) \to (a', b')$ is called a *transition* and $\delta_1(a, b) = a'$ and $\delta_2(a, b) = b'$ are defined.

A simplification is that all agents concurrently sense their environment, as a response to a global start signal, and each one of them receives some input symbol from X. For example, X could contain all possible barometric pressures that the agent's sensors can detect. Then all agents concurrently apply the input function I to their input symbols to obtain their initial state. In this manner, the *initial configuration* of the system is formed. A *configuration* in general, given a *population* of n agents, is any string from Q^n, in which the i-th symbol is the state of agent i, $1 \leq i \leq n$ (by assuming an ordering on the agents).

Now the crucial part is that any interaction between the agents is possible and this is interpreted as some inherent nondeterminism of the system. Formally, the agents are organized in an interaction graph $G = (V, E)$, which is a directed graph without self-loops and multiple edges, and where V is a population of $|V| = n$ agents and E describes the permissible interactions. Angluin *et al.* in [1] and [9] modeled mobility via an *adversary scheduler* that is a black-box to the protocol and simply selects members of E to interact according to δ (all agents apply the same global transition function). The only, but necessary, restriction imposed on the scheduler is that it has to be *fair* so that it does not forever partition the network into non-communicating clusters and, as Aspnes and Ruppert cleverly observe in [14], to prevent the possibility of having agents interacting only at "inconvenient" times. An *execution* (which is any sequence of configurations) is said to be fair if there exists no configuration in the execution that appears infinitely often while some successor of it does not. A *computation* is an infinite fair execution.

Computations by definition do not halt. In fact, the definition itself captures the inherent inability of such systems to detect termination, which is mainly due to the *uniformity* and *anonymity* properties of population protocols. Uniformity requires protocol descriptions to be independent of n and anonymity that the set of agent states is enough small so that there is no room in it for unique identifiers. Instead, computations are required to stabilize to a correct common or distributed value. Angluin *et al.* used *functions on input assignments* in order to formalize the *specifications* of protocols. For example, a natural question in our example could be whether at least

one agent detects barometric pressure over some constant pressure c (which possibly partitions the pressures in low and increased). What we do expect from a protocol that solves this problem is to always *stabilize* (*converge*) in a finite number of steps to the correct answer. Since such a query has a binary range, making it a *predicate*, we want all agents to output 1 when the predicate is made true and 0 otherwise, which is a convention that we make for gathering the protocol' s output. Since our main aim here is to survey the computational power of the models under consideration, we can w.l.o.g. focus on predicates [14], although [1, 9] provided general definitions for functions and also proposed many other natural *output conventions*.

Formally, the input (also called an *input assignment*) to a PP $\mathcal{A}$ may be any $x \in X^*$, where X^* denotes the set of all strings that can be made by concatenating zero or more symbols from X. In fact, any such $x = \sigma_1\sigma_2 \dots \sigma_n$ can be the input to $\mathcal{A}$ when it runs on a population of size n. This is done by assuming an arbitrary ordering on the set of agents, which is hidded from the agents themselves. Then we can simply make the convention that agent i, $1 \leq i \leq n$, senses the input symbol σ_i. A specification for $\mathcal{A}$ is any predicate $p : X^* \to \{0, 1\}$ which can also be thought as its corresponding language $L_p = \{x \in X^* \mid p(x) = 1\}$.

The *transition graph* $T(\mathcal{A}, G)$ of a protocol $\mathcal{A}$ that is executed on an interaction graph G [9] is a directed graph whose nodes are all possible configurations and whose edges are all possible one-step transitions between those configurations.

Not all interaction graphs are allowed in most practical scenarios. For example, if no obstacles are present we can assume that the interaction graph is always complete, which is the most commonly studied case. On the other extreme, we could also consider only line graphs or even more complicated collections of restricted graphs. This network selection is captured by graph *universes* (also called *families*) which are sets of graphs. The set of all complete directed interaction graphs, that has been extensively studied, is such an example of graph universe. One simple way to think of graph universes is that they capture the interaction graphs on which our protocols are about to be executed.

A predicate p over X^* is said to be *stably computable* by the PP model in the graph universe $\mathcal{U}$, if there exists a PP $\mathcal{A}$ such that for any input assignment $x \in X^*$, any computation of $\mathcal{A}$, on any interaction graph from the set $\{G \in \mathcal{U} \mid |V(G)| = |x|\}$, beginning from the initial configuration corresponding to x reaches an output stable configuration in which all agents output $p(x)$.

Research first focused on complete interaction graphs. Note that the completeness of the graph implies that stably computable predicates have to be symmetric. A predicate p is called *symmetric* if for every $x \in X^*$ and any x' which is a permutation of x's symbols, it holds that $p(x) = p(x')$ (in words, permuting the input symbols does not affect the predicate's outcome). In [1, 9] the authors, among other results, established that any *semilinear predicate* or, equivalently [35], any predicate definable by first-order logical formulas in *Presburger arithmetic* [36], is stably computable by the PP model. Moreover, in [37, 33] it was proven that this inclusion is tight, thus, arriving to the following exact characterization for the computable predicates in complete graphs: *A predicate is stably computable iff it is semilinear.* This is a fairly small class, not including multiplication of variables, exponentiations, and many other important operations on input variables. In particular (according to [1]), each term in these predicates is a constant (nonnegative integer), a variable, or the sum of two terms, or the product of a constant and a term, or the integer quotient of a term and a nonzero constant, or the remainder of a term modulo a nonzero constant. Atomic expressions are formed from two terms and one of the predicates: $=, \leq, <, \geq, >$. For example, the predicate $((17N_1 \geq 3N_0) \wedge (4N_1 \leq N_0))$ expresses that the input assignments to be accepted are those in which the number of 1s is between 15% and 20% of the total population. Moreover, Delporte-Gallet *et al.* [38] showed that PPs can tolerate only $\mathcal{O}(1)$ crash failures and not even a single Byzantine agent [1]. These weaknesses were inevitable given that the PP model was on purpose designed to be minimalistic. We do not attempt a complete survey of the results concerning population protocols since there is already an excellent one [14]. Finally, for a step by step (less technical than the current article) introduction to population protocols and related models, including examples and exercises, the interested reader is referred to [40].

Another possibility is to allow the inputs to the population oscillate and only require that this oscillation ceases after some finite time. This is the *stabilizing inputs* variant of population protocols [41]. Here all agents are assumed to be initially in some *initial state* q_0 (i.e. no input function is defined) and the transition function is now of the form $\delta : (Q \times X) \times (Q \times X) \to Q \times Q$. Each agent is like having two components in its state, where the first one is its actual state from Q and the second one plays the role of an input port whose symbol may change arbitrarily between any two interactions but it will eventually stabilize. The output function is applied to the state components of the agents. In [41] it was shown that all semilinear predicates can be computed with stabilizing inputs, which, together with the above exact characterization for population protocols, implies that *any population protocol for fixed inputs can be adapted to work with stabilizing inputs.* Moreover, [37] also established that nothing more than semilinear predicates can be computed by the stabilizing inputs variant.

For some other relevant previous work, [1, 9] also proposed the *probabilistic population protocol* model, in which the scheduler selects randomly and uniformly the next pair

[1] A Byzantine agent is an agent that has a Byzantine failure, that is, during any interaction it may pretend to be in any state (see e.g. [39] for a lot more on Byzantine failures and how they are handled in classical distributed systems).

to interact. Some recent work has concentrated on performance, supported by this random scheduling assumption (see e.g. [42]). [43, 44] considered a huge population hypothesis (population going to infinity), and studied the dynamics, stability and computational power of probabilistic population protocols by exploiting the tools of continuous nonlinear dynamics. Moreover, several extensions of the basic model have been proposed in order to more accurately reflect the requirements of practical systems. In [41], Angluin *et al.* also studied what properties of restricted interaction graphs are stably computable by the population protocol model and gave protocols for some of them. Recently, Bournez *et al.* [45] investigated the possibility of studying population protocols via game-theoretic approaches.

3.1. Mediated Population Protocols

In [10] (see also [46] for an extensive presentation of the results discussed in this section) the authors considered the following enhancement. They made the assumption that each interaction link is also a finite-state agent that only communicates with the agents it joins. In particular, whenever any two agents u and v interact via the edge $e = (u, v)$, u and v read the state of e and their own states and update all of them according, again, to some global transition function. Now, it is like the agents are capable of leaving small pieces of pairwise information that will be available during their subsequent interactions. The authors named their model *Mediated Population Protocol* (MPP) model.

Mediated Population Protocol $\mathcal{A}$ (MPP $\mathcal{A}$): A mediated population protocol $\mathcal{A}$ is a 7-tuple $\mathcal{A} = (X, Y, Q, S, I, O, \delta)$, where X, Y, Q (it is now called set of *agent states*), I, and O are as in the PP model, S is a finite set of *edge states*, and the transition function is now of the form $\delta : Q \times Q \times S \to Q \times Q \times S$.

As you may have noticed, there is no input to the edges (although such an input function was defined in [10]). We follow here the simplified definition given in [47] and consequently make the assumption that there exists some *edge initialization function* $\iota : E \to S$, that specifies the initial state of each edge. Note that ι is not part of the protocol but generally models some preprocessing on the network that has taken place before the protocol's execution. We focus on complete communication graphs and initially identical edges, that is, $\iota(e) = s_0$ for all $e \in E$. MPS is the class of all predicates that are stably computable by the so called "Symmetric" MPP model (SMPP).

Theorem 1 ([10]). *The class of semilinear predicates is a proper subset of MPS.*

Proof Idea. The PP model is a special case of the MPP model and, thus, any semilinear predicate is stably computable by the SMPP model. Consider now the non-semilinear predicate $(N_c = N_a \cdot N_b)$ which is true iff the number of cs in the input is the product of the number of as and the number of bs. By exploiting the fact that in complete graphs the number of edges leading from a-agents to b-agents is equal to the product, it is easy to devise a SMPP protocol $\mathcal{A}$ with *stabilizing states* (i.e. whose states eventually stop changing) that provides the following semilinear guarantee: If $N_c \neq N_a \cdot N_b$ then at least one agent remains in some state from some $Q' \subset Q$, otherwise no such state remains. Finally, compose $\mathcal{A}$ with a SMPP $\mathcal{B}$ that stably computes with stabilizing inputs the above guarantee to obtain a SMPP that stably computes the non-semilinear predicate $(N_c = N_a \cdot N_b)$. □

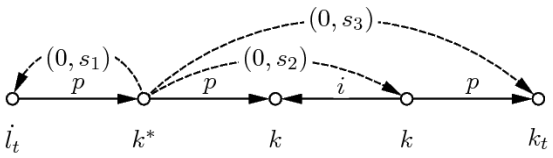
The above result of [10] was a first indication that MPP protocols in general can exploit the extra information stored on the edges in order to compute more complicated predicates. The following theorem shows that this information increases the computational power to the maximum possible degree, which was hardly expected at the beginning. Generally, by $NSPACE(f(n))$ we denote the class of all languages decided by some $\mathcal{O}(n^2)$ space nondeterministic Turing machine (a deterministic Turing machine is abbreviated *TM*, while a nondeterministic one is abbreviated *NTM*, according e.g. to Sipser [48]), where n here denotes the size of the input to the machine.

Theorem 2 ([10, 47]). *A predicate is in MPS iff it is symmetric and is in $NSPACE(n^2)$.*

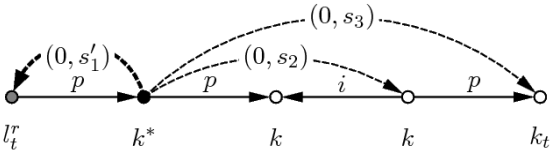
Proof Idea. The "only if" part is easy. Any predicate in MPS is obviously symmetric and additionally we can perform in $\mathcal{O}(n^2)$ space a nondeterministic search on the transition graph of the SMPP that stably computes the predicate.

The sufficiency of the conditions is somewhat more complicated. We have to show that for all symmetric languages $L \in NSPACE(n^2)$ there exists a SMPP that stably computes p_L, defined as $p_L(x) = 1$ iff $x \in L$. It is easy to see that p_L is symmetric iff L is symmetric. The idea is to organize the agents into a spanning line subgraph of the interaction graph. To do that, the agents begin to form small line graphs that in the sequel are merged together and are expanded to isolated nodes. When this process ends, the edges of the spanning line graph will be active and all other $\mathcal{O}(n^2)$ edges will be inactive. Now the network can operate as a Turing machine of $\mathcal{O}(n^2)$ space by using the agents as the control units and the inactive edges as the cells (see Figure 1). Whenever the inactive edges of some agent are exhausted it passes control (via some active edge) to its neighbor on the spanning line graph. By also exploiting the nondeterminism inherent in the interaction pattern the agents can simulate the nondeterministic TM that decides L. Note that, since the agents cannot detect termination of the spanning line graph construction process, any time that the structure changes they reinitialize their computation in a systematic manner, so that reinitialized agents do not communicate with non-reinitialized ones, and by exploiting a backup of their input that is maintained throughout the computation. The final reini-

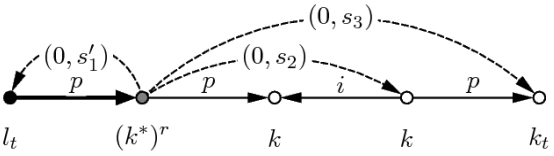
tialization happens when the spanning line graph is formed an then the simulation is executed correctly. □



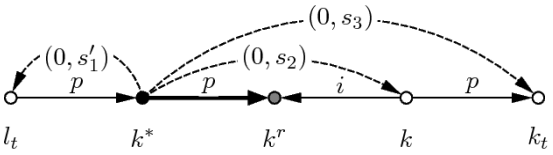
(a) The agent in k^* controls now the simulation.



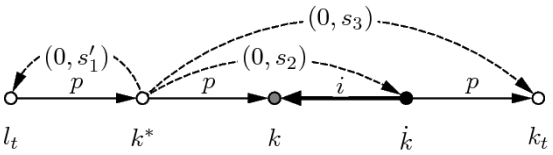
(b) A step of the simulation is executed on the inactive edge. The TM says "right" so k^* must next run the simulation on the first inactive edge to the right.



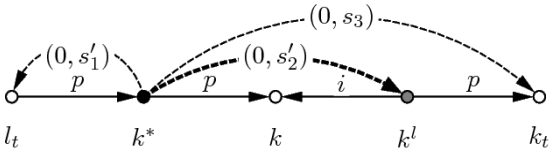
(c) Mark r travels to the right until it meets the first agent that has an incoming inactive edge from k^*.



(d) The mark still travels.



(e) The correct agent was found. The special dot mark will make the simulation run on the next inactive edge.



(f) A step of the simulation is executed. The TM says now "left" so the simulation must next use again the previous inactive edge.

Figure 1: An example of simulating a $\mathcal{O}(n^2)$-space TM. The simulation is performed on the second component of the inactive edges. The bold edge indicates the pair that has just interacted. The black agent is the initiator and the grey the responder.

3.2. Communicating Machines

Another natural enhancement of the population protocol model is to equip the agents with more memory and to enable them operate as Turing machines instead of being automata. For example, a natural question could be: "What can such a model compute when each agent's memory is logarithmic in the population size?". First of all, the *communicating machines* assumption is perfectly consistent with current technology (cellphones, iPods, PDAs, and so on). Moreover, *logarithmic* is, in fact, *extremely small*. For a convincing example, it suffices to mention that for a population consisting of 2^{266} agents, which is a number greater than the number of atoms in the observable universe, it is only required for each agent to have 266 cells of memory (while small-sized flash memory cards nowadays exceed 16GB of storage capacity)! This kind of questions engaged the author's attention in [11] who provided a general model independent of space bounds, called the PM model (standing for Passively mobile Machines).

Passively mobile Machines protocol $\mathcal{A}$ (PM $\mathcal{A}$): Formally, a PM protocol $\mathcal{A}$ is a 6-tuple $\mathcal{A} = (X, \Gamma, Q, \delta, \gamma, q_0)$ where X, Γ and Q are all finite sets and

1. X is the *input alphabet*, where $\sqcup \notin X$ ($\sqcup$ is the *blank symbol*),
2. Γ is the *tape alphabet*, where $\sqcup \in \Gamma$ and $X \subset \Gamma$,
3. Q is the set of *states*,
4. $\delta : Q \times \Gamma^4 \to Q \times \Gamma^4 \times \{L, R\}^4 \times \{0, 1\}$ is the *internal transition function*,
5. $\gamma : Q \times Q \to Q \times Q$ is the *external transition function* (or *interaction transition function*), and
6. $q_0 \in Q$ is the *initial state*.

Each agent is equipped with the following:

- A *sensor* in order to sense its environment and receive a piece of the input (which is an input symbol from X).
- Four read/write *tapes*: the *working tape*, the *output tape*, the *incoming message tape* and the *outgoing message tape*. We assume that all tapes are bounded to the left and unbounded to the right.
- A *control unit* that contains the state of the agent and applies the transition functions.
- Four *heads* (one for each tape) that read from and write to the cells of the corresponding tapes and can move one step at a time, either to the left or to the right.
- A binary *working flag* either set to 1 meaning that the agent is *working* internally or to 0 meaning that the agent is *ready* for interaction.

Initially, each agent receives an input symbol from X which is written on the leftmost cell of its working tape. The working flag of all agents is initially set to 1 and their initial state is q_0. Each agent may perform some internal computation by using the global (common for all agents)

internal transition function δ. This computation may include modifications on the tapes, state changes, and, finally, altering the working flag. When the working flag becomes 0, the agent does no longer modify it and is waiting for an interaction. Interactions that take place while the agent is working internally, have no effect. When two agents that are both waiting for interaction interact the external transition function γ is applied, the states of the agents are updated accordingly, and the outgoing message of each agent is copied to the incoming message tape of the other (from the leftmost cell and leaving all remaining cells blank). This operation is called the *message swap*. Moreover, interactions are assumed to be atomic operations.

Note that the PM model *preserves uniformity*, because X, Γ, and Q are all finite sets whose cardinality is independent of the population size. Thus, protocol descriptions have also no dependence on the population size. Moreover, PM protocols are *anonymous*, since there is no room in the state of the agents for unique identifiers (though here there is plenty of room on the tapes to create such ids).

The authors in [11] focused again on complete interaction graphs and defined the complexity class $PMSPACE(f(n))$ consisting of all predicates that are stably computable by a PM protocol that uses $\mathcal{O}(f(n))$ space in every agent (and in all of its tapes). Again, an immediate observation is that only symmetric predicates can be stably computable by the PM model on complete graphs. Since they were mainly interested in computations space-bounded by a logarithm on the population size, they distinguished the class $PLM \equiv PMSPACE(\log n)$ and defined a *PALOMA* protocol (standing for PAssively mobile LOgarithmic space MAchines) as a PM protocol that always uses $\mathcal{O}(\log n)$ space. Then the authors gave an exact characterization for PLM by proving the following more general theorem.

Theorem 3 ([11]). *For every function $f(n) = \Omega(\log n)$, a predicate is in $PMSPACE(f(n))$ iff it is symmetric and is in $NSPACE(nf(n))$.*

Proof Idea. Again the "only if" part is not hard. In fact, this part holds for any function and not only those that are at least $\log n$. $p \in PMSPACE(f(n))$ means that there exists a PM protocol $\mathcal{A}$ using $\mathcal{O}(f(n))$ space in every agent that stably computes p. Thus, each agent configuration can be represented in $\mathcal{O}(f(n))$ space and each population configuration (consisting of the agent configurations of all agents) in $\mathcal{O}(nf(n))$ space. Then the NTM simply performs, as in Theorem 2, a nondeterministic search on the transition graph of protocol $\mathcal{A}$ by always storing at most one population configuration and decides the language $L_p = \{x \in X^* \mid p(x) = 1\}$.

For the other part, take any symmetric language $L \in NSPACE(\, nf(n))$. The predicate p_L defined as $p_L(x) = 1$ iff $x \in L$ is also symmetric. Obviously, we must present a PM protocol that stably computes p_L by using $\mathcal{O}(f(n))$ space in every agent. Assume that the PM model was equipped with the unique consecutive ids $\{0, 1, \ldots, n-1\}$. In this case, we could line up all agents and make them simulate the NTM $\mathcal{N}$ that decides L. In order to ensure that no agent exceeds the desired space bound, we could run the simulation in a modular way, that is, the first n cells of $\mathcal{N}$ correspond to the first cells of the n agents, and generally, the cells $kn+1, \ldots, (k+1)n$ of $\mathcal{N}$, $k \geq 0$, correspond to the $(k+1)$th cells of all agents. In this manner, the $\mathcal{O}(nf(n))$ space of $\mathcal{N}$ is evenly divided between n agents and this ensures that each agent uses $\mathcal{O}(f(n))$ space. Moreover, note that the unique ids consume $\mathcal{O}(\log n)$ space and since $f(n) = \Omega(\log n)$ there will always be enough space to store them. To simulate the nondeterminism of $\mathcal{N}$ the idea is to make a nondeterministic search on $\mathcal{N}$'s computation tree. The nondeterminism of the search stems from the inherent nondeterminism of the interaction pattern. Whenever the simulation must take a nondeterministic transition, the corresponding agent takes an ineffective interaction only to read the id of the other agent and this id is used to determine the nondeterministic choice to be made. Whenever the search reaches a rejecting node it is being restarted, beginning another nondeterministic search from the root. Obviously, the fairness of the execution guarantees that if there exists an accepting node it will be eventually found, otherwise the simulation will forever reject. Call the described protocol $\mathcal{B}$.

One important point remains: PM protocols do not have unique ids. However, at the space cost of $\mathcal{O}(\log n)$ PM protocols can create those ids. Initially, all agents have the same id 0. Whenever two of them meet, one of them increases its id by one. It is not so hard to see that this process eventually ends with a correct assignment of the unique ids $\{0, 1, \ldots, n-1\}$. A difficulty is that the agents cannot detect termination of this process, because, otherwise, it is easy to show that termination could have been erroneously detected in a subset of the population. So, we actually do not know when to start executing the protocol $\mathcal{B}$ that makes use of these ids. Only one option remains: whenever some id is modified all agents get informed of this event and start simulating $\mathcal{B}$ from the beginning. This *iterative reinitiation technique* (used also in [34]) guarantees that when the last id-modification takes place, when the agents have just obtained the correct ids, all agents will restart $\mathcal{B}$'s execution for the last time and $\mathcal{B}$ will be executed without further interruptions and as if the ids were provided to it from the beginning. Thus, it will correctly simulate $\mathcal{N}$ and will stably compute p_L. □

As PLM is by definition $PMSPACE(\log n)$, we can conclude that:

Corollary 1 ([11]). *PLM is equal to the class of all symmetric predicates in $NSPACE(n \log n)$.*

4. The Sensor Field Model

In [30] the authors propose a general model capturing some characteristic differences of sensor networks. A sensor field is composed of a kind of devices that can communicate one to the other and also to the environment. In this study the authors assume that the devices and the communication links do not appear and disappear during the computation. Furthermore, those devices synchronize at barriers marking rounds, in a way similar to the BSP model [31]. During a computation round, a device access the received messages and data provided by the environment, performs some computation, and finally sends messages to its neighbors and to the environment. Those are the fundamental features of the *Static and Synchronous Sensor Field* (SSSF) model. The model allows the definition of complexity measures like latency, round duration, message number or message length among others. In this setting it is possible to formulate a general and natural definition of sensing problems by means of input/output data streams. The model can be seen as a non-uniform computational model in the sense that it is easy to introduce constraints to all or some of the devices of the sensor field and relate it to classic complexity classes.

As described before, the general sensing setting considered in this model is described by the following elements: *the observers* and the *the phenomenon*. The phenomenon is the entity of interest to the observers that is being sensed and potentially analyzed by the sensor network. Multiple phenomena may be concurrently under consideration in the same network.

Depending on the information required by the observers from the environment, different kind of problems of interest can be formulated as sensing problems. Those problems have to be *solved* on a decentralized network composed of sensing units and other elements. We have to re-examine not only the meaning of succesful computation but also to examine different performance metrics to measure the efficiency of the computational solutions given in this new model. The vision of the cycle of problem solving by networked sensors correspond to the schema given in Figure 2. This schema discretizes information in two ways. First the environment is sampled only on a discrete set of locations and second the measures taken by the sensor are digitalized to the corresponding precision. We face a dual problem in analyzing correctness and performance. On one side we have a computational problem to be solved on a particular network (or a family of networks). On the other hand we have to asses whether the observation of the phenomenon is valid. It is clear that both tasks will require different analysis tools. The computational models focus on the first problem. Now we describe the main components of a sensor field.

A *communication graph* is a directed graph $G = (N, E)$ where N is the set of nodes and E is the set of edges, $E \subseteq N \times N$. Unless explicitly stated we assume that N has n nodes that are enumerated from 1 to n and m edges. Each

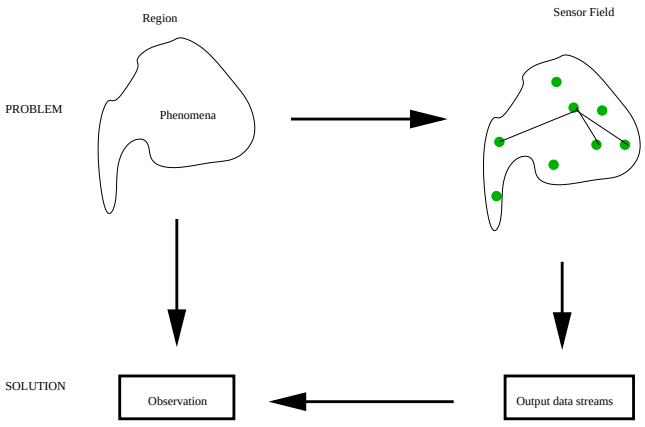


Figure 2: Problem solving with sensor fields.

node k is associated to a device, let us say to device k, that has access to the k-th data stream. Each edge $(i, j) \in E$ specifies that device i can send messages to device j or what is the same, device j can receive messages from device i. Given a device k let us denote by $I(k) = \{i \mid (i, k) \in E\}$ the set of neighbors from which device k can receive data items and by $O(k) = \{j \mid (k, j) \in E\}$ the set of neighbors to which device k can send data. Let $in_k = |I(k)|$ and $out_k = |O(k)|$ be the in and out degrees of node k. Set $in_G = \max_{k \in N} in_k$ and $out_G = \max_{k \in N} out_k$. We use d_G to denote the diameter of the graph G.

A *Static Synchronous Sensor Field* consists of a *set of devices* and a *communication graph*. The communication graph specifies how the devices communicate one to the other. For the moment and without loose of generality, we assume that all devices are sensing devices that can receive information from the environment and send information to the environment. Since the model we consider is static we assume that the edges are the same during all the computation time. Moreover each device executes its own process, communicates with their neighbors (devices associated to adjacent nodes) and also with the environment. All the devices work in a synchronous way, at the begining of each round they receive data from their neighbors and from the environment, then they apply their own transition function changing in this way their actual configuration and finish the round sending data to their neighbors and to the environment. Let us describe in detail the main components of the Static Synchronous Sensor Field.

Static Synchronous Sensor Field $\mathcal{F}$ (SSSF $\mathcal{F}$): Formally we define a Static Synchronous Sensor Field $\mathcal{F}$ by a tuple $\mathcal{F} = (N, E, U, V, X, (Q_k, \delta_k)_{k \in N})$ where

- $G_{\mathcal{F}} = (N, E)$ is the communication graph.
- U is the alphabet of data items used to represent the input data streams that can be received from the environment.
- V is the alphabet of items used to represent the out-

put data streams that can be send to the environment.

- X is the alphabet of items used to communicate each device to the other devices. Each $m \in X^*$ is called message or packet. $U, V \subseteq X$. We denote by data items the elements of alphabets U and V and by communication items (or items) the elements of X.

- (Q_k, δ_k) defines for each device associated to a node $k \in N$ (device k) its set of local states and its transition function, respectively.

The *local computation of each device k in $\mathcal{F}$* is defined by (Q_k, δ_k) and depends on the communication with its neighbors and with the environment. Q_k is a (potentially infinite) set of local states and δ_k is a transition function. A state codifies the values of some local set of variables (ordinary program variables, message buffers, program counters ...) and all what is needed to describe completely the instantaneous configuration of the local computation. The transition function δ_k that depends on its local state $q_k \in Q_k$ as well as on:

- the communication items received by device k from devices $i \in I(k)$,

- the data item that device k receives as input from the environment,

- the communication items sent by device k to devices $j \in O(k)$,

- and the data item that device k sends to the environment.

The transition function is defined as

$$\delta_k : Q_k \times (X^*)^{in_k} \times U \longrightarrow Q_k \times (X^*)^{out_k} \times V.$$

The meaning of

$$\delta_k(q_k, (x_{ik})_{i \in I(k)}, u_k) = (q'_k, (y_{kj})_{j \in O(k)}, v_k)$$

is that if device k of $\mathcal{F}$ is in its local state $q_k \in Q_k$, receives $x_{ik} \in X^*$ from each of its neighbors $i \in I(k)$, and receives the input data item $u_k \in U$ from the environment, then in one computation step device k changes its local state to $q'_k \in Q_k$, sends $y_{kj} \in X^*$ to each of its neighbors $j \in O(k)$ and outputs $v_k \in V$ to the environment. In the case that device k does not send any value, we denote this 'no value' or 'does not care' by the special symbol $\perp$. For any device k, let q_k^0 be the initial local state. For any $t \geq 1$, *the t-th computation round of device k* is described as follows: If the local state of device k is q_k^{t-1}, and it receives $(x_{ik}^t)_{i \in I(k)}$ from its input neighbors, u_k^t from the environment and $\delta_k(q_k^{t-1}, (x_{ik}^t)_{i \in I(k)}, u_k^t) = (q_k^t, (y_{kj}^t)_{j \in O(k)}, v_k^t)$ then device k changes its local state from q_k^{t-1} to q_k^t, sends $(y_{kj}^t)_{j \in O(k)}$ to its output neighbors and v_k^t to the environment.

A *computation* of $\mathcal{F}$ is a sequence

$$\mathbf{c}^0, \mathbf{d}^1, \mathbf{c}^1, \mathbf{d}^2, \ldots, \mathbf{c}^{t-1}, \mathbf{d}^t, \mathbf{c}^t, \ldots,$$

eventually infinite, where $\mathbf{c}^0 = (q_k^0)_{k \in N}$ is the n-tuple of the initial local states of the n devices, and for each $t \geq 1$, $\mathbf{c}^t = (q_k^t)_{k \in N}$ is the n-tuple of the local states after t computation rounds. The tuple $\mathbf{d}^t = (d_k^t)_{k \in N}$ represents the input/output data of the t-th computation round (i.e. the transition from round $t-1$ to round t). In particular, for device k the input/output data of the t-th round is represented by $d_k^t = ((x_{ik}^t)_{i \in I(k)}, u_k^t, (y_{kj}^t)_{j \in O(k)}, v_k^t)$. Note that device k receives $(x_{ik}^t)_{i \in I(k)}$ from its neighbors, $x_{ik}^t = y_{ki}^{t-1}$ receives u_k^t from the environment, changes its state from q_k^{t-1} to q_k^t, sends $(y_{kj}^t)_{j \in O(k)}$ to its neighbors and sends v_k^t to the environment.

The *stream behavior of a computation*

$$\mathbf{c}^0, \mathbf{d}^1, \mathbf{c}^1, \mathbf{d}^2, \ldots, \mathbf{c}^{t-1}, \mathbf{d}^t, \mathbf{c}^t, \ldots$$

of $\mathcal{F}$ is defined as $(\mathbf{u}, \mathbf{v})$ where $\mathbf{u} = (u_k)_{k \in N}$ is the tuple composed by the input data streams of each device k, $u_k = u_k^1 u_k^2 \ldots u_k^t \ldots$ and $\mathbf{v} = (v_k)_{k \in N}$ is the tuple composed by the output data stream of each device $v_k = v_k^1 v_k^2 \ldots v_k^t \ldots$ Notice that this information can be extracted from the computation $\mathbf{c}^0, \mathbf{d}^1, \ldots, \mathbf{c}^{t-1}, \mathbf{d}^t, \mathbf{c}^t, \ldots$. Thus the sensor field $\mathcal{F}$ outputs the tuple of output data streams $\mathbf{v} = (v_k)_{k \in N}$ given the tuple of input data streams $\mathbf{u} = (u_k)_{k \in N}$ or what is the same, $\mathbf{v}[1,t]$ given $\mathbf{u}[1,t]$ for each $t \geq 1$.

We define the function $f_{\mathcal{F}}$ associated to the stream behavior of $\mathcal{F}$ as follows: Given any pair of tuples of data streams $\mathbf{u}$ and $\mathbf{v}$ and any $t \geq 1$, $f_{\mathcal{F}}(\mathbf{u}[1,t]) = \mathbf{v}[1,t]$ if and only if the sensor field $\mathcal{F}$ computes $\mathbf{v}[1,t]$ given $\mathbf{u}[1,t]$.

Function computed by $\mathcal{F}$: A function f (defined on data streams) is *computed by a sensor field $\mathcal{F}$ with latency d* if for all (appropriate) tuple of data streams $\mathbf{u}$, and for all $t \geq 1$, $f_{\mathcal{F}}(\mathbf{u}[1,t+d])[t+d] = f(\mathbf{u}[1,t])[t]$. That is the SSSF outputs at time $t+d$ the t-th element of f. We say that f is *computed by a sensor field $\mathcal{F}$* if there exists d for which f is computed by $\mathcal{F}$ with latency at most d.

Note that $\mathbf{u}$ and $\mathbf{v}$ have in general infinite length. In order to express formally the behavior of a SSSF we consider all the finite prefixes of the input stream ($\mathbf{u}[1,t]$) and those of the output stream ($\mathbf{v}[1,t]$). However, take into account that each sensor will output only one data item ($\mathbf{v}[t]$) per round.

The computational resources used by a sensor to compute a function of this kind are the following. For each device and computation round we can measure

- *Time*. The number of operations performed in the given round of the device. This is a rough estimation of the "physical time" needed to input data, receive information from other sensor, compute, send information and output data.

- *Space*. The space used by the device in such computation round.

- *Message Length.* The maximum number of data items of a message sent by the device in such computation round.

- *Number of messages.* The maximum number of messages sent by the device in such round.

We consider the following worst case complexity measures taken over any device and computation round of a sensor field $\mathcal{F}$:

- *Size*: The number of nodes or devices of the communication graph G.
- *Time* ($\mathcal{T}$): The maximum time used by any device in any of its rounds.
- *Space* ($\mathcal{S}$): The maximum space used by any device of in any of its rounds.
- *MessageLength* ($\mathcal{L}$): The maximum message length of any device of in any of its rounds.
- *MessageNumber* ($\mathcal{M}$): The maximum number of messages sent by any device in any of its rounds.

In general we analyze these complexity measures with respect to the *Size* of the communication graph which usually will coincide with the number n of data streams, we denote by $\mathcal{T}(n)$ the *Time*, by $\mathcal{S}(n)$ the *Space*, by $\mathcal{L}(n)$ the *MessageLength* and by $\mathcal{M}(n)$ the *MessageNumber*.

Computational problems that are susceptible of being solved by a sensor field are any of the family of problems defined in Section 2. Recall that all those problems can be stated as a sensing problem.

Sensing Problem Π: Given an n-tuple of data streams $\mathbf{u} = (u_k)_{1\leq k\leq n}$ for some $n \geq 1$, compute an m-tuple of data streams $\mathbf{v} = (v_k)_{1\leq k\leq m}$ for some $m \leq n$ such that $R_\Pi(\mathbf{u}[1,t], \mathbf{v}[1,t])$ is satisfied for every $t \geq 1$.

Therefore it remains to formalize the notion of solved problem.

Problem Solved by $\mathcal{F}$: A sensor field $\mathcal{F}$ *solves problem* Π *with latency* d if for every pair of data streams $\mathbf{u}$ and $\mathbf{v}$, and every $t \geq 1$, if $f_\mathcal{F}(\mathbf{u}[1,t]) = \mathbf{v}[1,t]$ then $R_\Pi(\mathbf{u}[1,t], \mathbf{v}[1+d, t+d])$. A sensor field $\mathcal{F}$ *solves the problem* Π if there is a d such that $\mathcal{F}$ solves problem Π with latency d.

This definition introduces an additional parameter *the latency*. This latency specifies the number of time steps that we have to wait before the output data stream coincides with the problem's output data stream.

To compare formally the computational power of sensor fields with some classical language clases we need to define the decisional version of $f_\mathcal{F}$.

Language associated to $\mathcal{F}$: Let $\mathcal{F}$ be a SSSF and let $f_\mathcal{F}$ be the function associated to the behavior of $\mathcal{F}$. We define the language associated to the behavior of $\mathcal{F}$, denoted by $L(\mathcal{F})$, is the set

$$\{\langle \mathbf{u}[1], \mathbf{v}[1], \ldots, \mathbf{u}[t], \mathbf{v}[t]\rangle \mid t \geq 1 \text{ and } f_\mathcal{F}(\mathbf{u}[1,t]) = \mathbf{v}[1,t]\}.$$

4.1. Solving sensing problems

We report here some results on the Average Monitoring problem and the Alerting problem introduced in Section 2 together with some additional results reported in [30].

4.1.1. Average Monitoring

The study of the requirements of a SSSF for solving the Average Monitoring problem can be divided in two parts. The first one provides lower bounds on some parameters and the second providing sensor fields giving upper bounds. Those algorithms provide matching upper bounds for some particular topologies.

Lower Bounds. In order to be able to state lower bounds, we make an additional assumption: all the sent messages are formed only by tuples of input data items (without compression). An easy argument shows the following lower bound.

Lemma 1 ([30])**.** *A SSSF $\mathcal{F}$ with communication graph G solving the Average Monitoring problem requires at least latency d_G.*

In the following results we assume, in addition, that along the whole computation the flow of packets from node i to node j, for any $i, j \in N$, follows a fixed path $p_{i,j}$. Thus the algorithm uses a fixed *communication pattern* $P = (p_{i,j})_{i,j\in N}$. Let $\beta_P(k)$ be the out-degree of node k in the subgraph G' of G formed by the paths in P that start at k and have length d_G. Set $\beta(G, P) = \max_{k\in N} \beta_P(k)$. Observe that those subgraphs are critical in terms of the delivery of packets in d_G rounds. It is easy to show the following lower bound on *MessageNumber*.

Lemma 2 ([30])**.** *Let $\mathcal{F}$ be a SSSF, with communication graph G and communication pattern P, solving the Average Monitoring problem with latency d_G. It holds that for any round $t > d_G$, there is a device sending at least $\beta(G)$ packets simultaneously.*

Taking into account that the different communication flows must be pipelined along paths with critical length the following lower bound on *MessageLength* can be established.

Lemma 3 ([30])**.** *Let $\mathcal{F}$ be a SSSF, with communication graph G and communication pattern P, solving the Average Monitoring problem with latency d_G. Then, if $k_1, \ldots, k_{d_G+1}$ is the fixed communication path used by devices $k_1, \ldots, k_{d_G}$ to send their data to k_{d_G+1} in P, there is a round $t_0 > d_G$ such that for any round $t > t_0$, there is a device receiving a message composed of at least d_G data items.*

Algorithms. We first sketch a generic SSSF with optimal latency provided that the communication graph is strongly connected. In general, this algorithm is not optimal in *Space*, *MessageLength* and *MessageNumber*, however when the topology of the communication graph is

known in advance, it is possible to obtain SSSF s with specific topologies that optimize such parameters. In what follows, we assume that every device in the SSSF is aware of the total number of devices n and the diameter d of the communication graph.

The algorithm is based on simple flooding. Each device keeps a table M of size $d \times n$ of data items. That is forwarded at each time step to the neighbors. At each time step the information is updated, taking into account the actual measurement at the node and the data received from the neighbors. This simple algorithm provides the following upper bounds.

Lemma 4 ([30]). *Let G be a strongly connected communication graph with n nodes. There is a SSSF $\mathcal{F}$ with communication graph G solving the Average Monitoring problem with latency d_G, $\mathcal{T}(n) = \mathcal{O}(n\, d_G(in_G + out_G))$, $\mathcal{L}(n) = \mathcal{O}(n\, d_G \log n)$, $\mathcal{S}(n) = \mathcal{O}(n\, d_G \log n)$ and $\mathcal{M}(n) = out_G$.*

Algorithms with optimal latency. When the topology of the communication graph is known it is possible to improve the generic algorithm to obtain optimal algorithms provided latency is kept at its minimum. The lower bounds follow from Lemmas 1, 2, and 3 taking into account the considered topologies.

Theorem 4 ([30]). *The Average Monitoring problem can be solved with latency d_G and optimal MessageNumber and MessageLength by SSSFs whose communication graph are bidirectional cliques, oriented rings or balanced binary trees, respectively.*

Improving the message length. By data aggregation and allowing a larger latency, it has been possible to improve the *MessageLength*. In this case, messages are no longer tuples of data items but sums of data items. The synchronization needed to compute the right sums forces an increment on the latency.

Theorem 5 ([30]). *The Average Monitoring problem can be solved with latency $2n - 1$, $\mathcal{T}(n) = \Theta(n)$, $\mathcal{S}(n) = \Theta(n \log n)$, $\mathcal{L}(n) = \Theta(\log n)$ and $\mathcal{M}(n) = \Theta(1)$ by a SSSF in which the communication network is an oriented ring.*

4.1.2. Alerting

The Alerting problem can be solved in constant memory SSSF by the following algorithm. Initially all the nodes are in a non-alert state. At any round, if an unalerted device receives an alert message or reads a data that provokes an alert changes it state to alert and sends an alert message. An alerted device, different from device 1 does nothing. Device one upon achieving the alert state outputs 1 at each round. This gives the following bounds.

Lemma 5 ([30]). *Let G be a communication graph in which there is a path from any node to node 1. There is a SSSF $\mathcal{F}$ with communication graph G solving the Alerting problem with latency bounded by d_G, $\mathcal{T}(n) = \Theta(1)$, $\mathcal{S}(n) = \Theta(1)$, $\mathcal{L}(n) = \Theta(1)$ and $\mathcal{M}(n) = \mathcal{O}(out_G)$.*

4.1.3. Trading space/time for size

In general, we can say that by restricting the memory capacity of each device to be a constant w.r.t. the total number of devices then the kind of problems solved by these SSSFs are not more difficult than the ones in DSPACE($\mathcal{O}(n+m)$).

Theorem 6 ([30]). *Let $\mathcal{F}$ be a constant space SSSF. Then, the language $L(\mathcal{F}) \in DSPACE(\mathcal{O}(n+m))$.*

It is natural to ask whether an additional amount of nodes in the communication graph in which the attached devices participate in the computation but do not play any active role in sensing. In such a network we have a communication graph with S nodes and we want to solve a problem that involves only $n < S$ input data streams.

On particular topologies the additional nodes together with pipeline allows to obtain constant time sensor fields for solving the Average Monitoring. The particular topology is a balanced communication tree in which it is assumed that there are n sensing devices placed on the leaves of the tree, edges to leaves are replaced by paths in such a way that all the leaves are at the same distance to the root. Thus, the tree has depth $\mathcal{O}(\log n)$ and n leaves. In such a network an algorithm with two flows can be considered. In the bottom-up flow each node receives from its children the average of the data at the subtree leaves, together with the number of leaves, and computes its corresponding values to be sent to its parent. The top-down computation is initiated by the root that computes the average value which flows to the leaves. The analysis is summarized as follows.

Theorem 7 ([30]). *Let G be a balanced communication tree whose n leaves are sensing devices with constant space and whose internal nodes are non-sensing devices with space $\mathcal{O}(\log n)$. There is a SSSF $\mathcal{F}$ with communication graph G solving the Average Monitoring problem with latency d_G, $\mathcal{S}(n) = \mathcal{L}(n) = \mathcal{O}(\log n)$ and $\mathcal{T}(n) = \mathcal{M}(n) = \mathcal{O}(1)$.*

In the algorithm described above the nodes in the communication tree require different levels of internal memory, ranging from constant at the leaves to $\log n$ in the upper levels. The following result shows that by increasing the number of auxiliary nodes we can solve sensing problems with constant memory components in an adequate topology within constant time.

Let $\mathcal{P}$ be a property defined on U^n. We consider the following sensing problem:

Monitoring Problem for property $\mathcal{P}$: Given an n-tuple of data streams $u = (u_k)_{1 \le k \le n}$ for some $n \ge 1$, compute an n-tuple of data streams $v = (v_k)_{1 \le k \le m}$ such that $\mathbf{v}[t] = \mathcal{P}(\mathbf{u}[t])$ for every $t \ge 1$.

Any polynomially computable property can be decide by a uniform family of circuits with polynomial size. Furthermore those circuits can be assumed to be layered and to have bounded fan in and fan out by adding propagator gates. The communication network is formed by the

circuit with sensors attached to the corresponding inputs together with a communication tree that flows the result to the inputs. As the circuit is layered we can guarantee the pipelined flow of partial computations with constant time and memory within latency equal to the circuit's depth plus the tree depth. Thus, we have polynomial in n.

Theorem 8 ([30]). *Let $\mathcal{P}$ be a property defined on U^n computable in polynomial time. There is a constant space SSSF that solves the associated sensing problem in polynomial size and latency (with respect to n) with $\mathcal{S}(n) = \mathcal{T}(n) = \mathcal{L}(n) = \mathcal{M}(n) = \mathcal{O}(1)$.*

5. Sensor fields with constant memory per device

The study of sensor fields has been enlarged by considering a submodel, the *Constant Memory Static Synchronous Sensor Field* (CMSF) [49]. In this variation it is assumed that the nodes in the network have constant available space. In particular this property forces to do not have node ids. To keep open the possibility of a high degree per node, we consider a realistic communication model in which sending a message correspond to a *broadcast to all* the neighbors. This is a normal hypothesis in wireless networks. For the reception of messages we assume that the nodes can receive only a constant amount of messages. The received data is selected according to some rule from the colliding messages.

A *Constant Memory static synchronous Sensor Field* (CMSF) consists of a *set of devices* and a *communication graph*. The communication graph specifies how the devices communicate one to the other. We assume that all devices are sensing devices that can receive information from the environment and send information to the environment. Since the model we consider is static we assume that the edges are the same during all the computation time. All the devices have the same local program. They work in a synchronous way, at each time step they receive data from their neighbors and from the environment, apply the transition function changing in this way their actual configuration and send data to their neighbors and to the environment. To keep memory constant inside the node rather than forcing that all nodes have bounded degree we assume the following communication assumptions:

- When a node sends a message it can potentially reach any of its neighbors, that is they use a *broadcast to neighbors* primitive.
- When only one neighbor is sending a message, the node receives it.
- When more that one neighbor is sending a message, the node receives one of the sent messages selected arbitrarily (the arbitrary model).

Observe that the last rule could be replaced by other rules for example, the one with highest value (the priority model) or not receiving any message (collision model). So far there only generic results are in the arbitrary model.

Let us define formally the constant memory static synchronous sensor field as a tuple $\mathcal{F} = (N, E, U, V, W, Q, \delta, q_0)$ where

- $G_{\mathcal{F}} = (N, E)$ is the communication graph.
- U is the alphabet of data items used to represent the input data streams that can be received from the environment.
- V is the alphabet of items used to represent the output data streams that can be send to the environment.
- W is the alphabet of items used to communicate each device to the other devices. $U, V \subseteq W$. We denote by data items the elements of alphabets U and V and by communication items (or items) the elements of W. We assume that there is a special symbol $\perp \in W$ that represents the fact that no message has been sent/received.
- Q is a finite set of states.
- $\delta : Q \times U \times W \to Q \times V \times W$ a transition function.
- $q_0 \in Q$ is the initial state.

The local computation of any device in $\mathcal{F}$ is defined by (Q, δ) and depends on the items received from its neighbors and the data items read from the environment. The transition specifies a new state, the communication item to be sent, and the output data item. Observe than in contraposition with the SSSF model Q is now finite.

An interesting question is to analyze under which conditions CMSF can simulate population protocols [9] or mediated population protocols (MPP) [10]. For doing so it is required to understand the communication process that takes part during an interaction of the population protocol. This interaction results in a definition of an adequate communication graph that allows the information interchange required. Moreoever, the simulation needs a way to map any fair adversary for a population protocol into a fair input data stream for a sensor field that keeps the same set, as well as, the same order of interactions alive.

The simulation is shown by construction, for any given population protocol a corresponding sensor field is defined. Additionally, any potential scheduler will be mapped to a tuple of input data streams. Those input data streams will guarantee that in the simulation of an interaction that involves a node a as actuator and a node r as respondant, two nodes of the communication graph will be awaken by the environment reciving a symbol a or r during the duration of the information exachange.

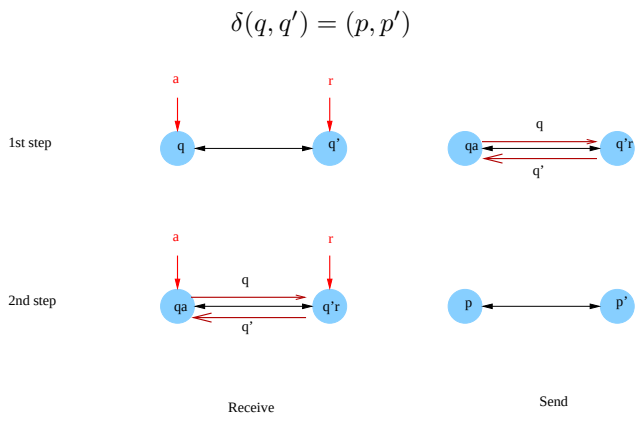


Figure 3: The two communication steps involved in a PP interaction

5.1. Population Protocols

In the case of population protocols, an interaction of two nodes involves the interchange of their states. From the communication point of view this requires two steps of communication (see Figure 3). In the first step two nodes will be awaken by the environment receiving the signal a or r during the simulation of an interaction that involving a as actuator and r as respondent. The remaining nodes do not receive any signal. In the second this two nodes interchange their state.

Observe that the interchange of information requires bidirectional communication even if the interaction is directed. The formal definition of the sensor field is the following.

Given a PP $\mathcal{P} = (X, Y, Q, I, O, \delta)$ that runs on interaction graph $G = (N, E)$, we define the SSSF $\mathcal{F} = \mathcal{F}(\mathcal{P}, G) = (N, E \cup E', U, V, W, Q', \delta', q_0)$ as follows:

- $E' = \{(u, v) \mid (v, u) \in E\}$.
- $U = X \cup \{a, r, \perp\}$, $V = Y$, and $W = Q \cup \{\perp\}$
- $Q' = Q \cup Q \times \{a, r\}$
- The transition function δ' is defined as follows:
  - $\delta'(q_0, x, \perp) = (I(x), \perp, \perp)$, for any x.
  - $\delta'(q, \perp, z) = (q, O(q), \perp)$, for any $q \in Q$, $z \in U$.
  - $\delta'(q, a, \perp) = ((q, a), O(q), q,)$, for any $q \in Q$.
  - $\delta'(q, r, \perp) = ((q, r), O(q), q)$, for any $q \in Q$.
  - $\delta'((q, a), a, q') = (\delta_1(q, q'), O(q), \perp)$, for $q \in Q$.
  - $\delta'((q, r), r, q') = (\delta_2(q', q), O(q), \perp)$, for $q \in Q$.

Where we use the projections of the transition function of the PP, that is $\delta(q, q') = (\delta_1(q, q'), \delta_2(q, q'))$. Observe that the transition function δ' implements the two communication rounds required by the interaction. The output is set to be the output corresponding to the last state in the population protocol to keep compatibility between the output data stream in the sensor field and the output of the population protocol.

The second step is to define a way to transform an input x and a scheduler S for $\mathcal{P}$ to a set of input data streams $u(x, S)$ to be used as input data stream by the sensor field. In this case the transformation is uniform to be able to expand one interaction two two timw steps.

The input data stream $u = u(x, S)$ is the following:

- For any node k, $u_k[1] = x_k$
- If the t-th interaction of the scheduler is (i, j) then
  - $u_i[2t, 2t + 1] = aa$
  - $u_j[2t, 2t + 1] = rr$
  - $u_\ell[2t, 2t + 1] = \perp\perp$, for $\ell \neq i, j$.

Observe that a symbol a on the input data stream codes the fact that the node participates in an interaction as actuator, r as a respondent, and $\perp$ indicates that the node does not participate in any interaction at step t. The repetition corresponds to the fact that an interaction requires two communication steps.

Theorem 9 ([49]). *Given a graph G and a population protocol $\mathcal{P}$ consider the CMSF $f(\mathcal{P}, G)$ defined above. For any scheduler S and input x consider the data stream $u(x, S)$ defined above. The t-th configuration of $\mathcal{P}$ on input x and scheduler S is identical to the $2t-1$-configuration of $f(\mathcal{P})$ on input data stream $u(x, S)$.*

5.2. Mediated Population Protocols

The simulation of a mediated population protocol by a sensor field can be done in two different ways. In the first one the network has no additional devices. The state of the edge participating in the interaction is assumed to be kept by the environment, and it is revealed to the agents during the interaction. As before the simulations consists of a construction of a sensor field and of a transformation of input and mediated scheduler to a tuple of input data streams.

Given a MPP $P = (X, Y, Q, S, I, O, \delta)$ that runs on interaction graph $G = (N, E)$, define the following SSSF $f(P, G) = (N, E', U, V, W, Q', \delta', q_0)$ as follows:

- $E' = E \cup \{(u, v) \mid (v, u) \in E\}$.
- $U = X \cup Y \cup \{a, r, \perp\}$
- $V = Y$ and $W = \{Q \cup \{\perp\}\}$
- $Q' = \{q_0\} \cup Q \cup (Q \times \{a, r\})$

The transition function δ' is defined as in the case of population protocols to keep the communication steps required by the interaction as depicted in figure (see Figure 4). For mediated population protocols an interaction requires two communication steps. Observe that in the first step the environment provides the role of the nodes in the interaction and in the second step the state of the edge participating in the interaction to the interested nodes. As

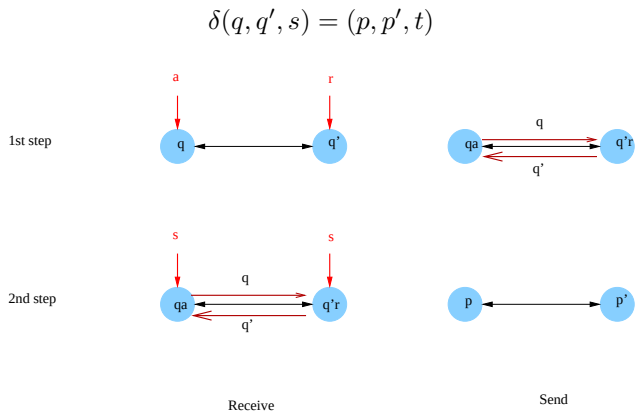


Figure 4: The two communication steps involved in a MPP interaction

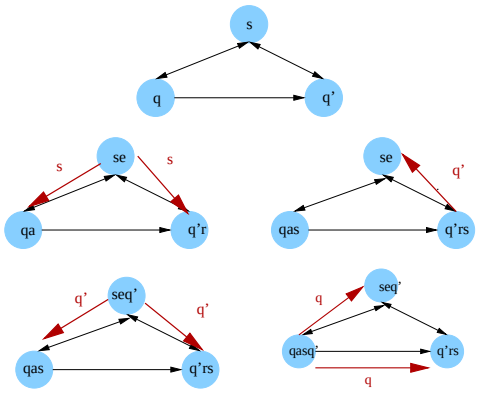


Figure 5: The four sending phases involved in a MPP interaction when edge nodes are added

before, the output data item is set to the output of the poulation protocol corresponding to the last seen state, and in the second step to the new state of the edge.

Next step is to define a way to transform a scheduler for the population protocol to a set of input data streams for the sensor field.

The input data stream $u = u(x, S)$ is the following:

- For any node k, $u_k[1] = x_k$
- If the t-th interaction of the scheduler is (i, j) and the actual state of the link (i, j) is $s \in S$ then
  - $u_i[2t, 2t+1] = as$
  - $u_j[2t, 2t+1] = rs$
  - $u_k[2t, 2t+1] = \perp\perp$, for any $k \neq i, j$.

Observe that in this simulation, we are assuming that the consistency of the edge states is preserved by the environment and that the edge state do not use network resources.

Theorem 10 ([49]). *Given a communication graph G and a mediated population protocol $\mathcal{P}$ consider the CMSF $f(\mathcal{P}, G)$ defined above. For any input x and mediated scheduler S, the t-th configuration of $\mathcal{P}$ on communication graph G, input x, and scheduler S is identical to the $2t-1$-configuration of $f(\mathcal{P}, G)$ on input data stream $u(G, x, S)$.*

There is another way to perform the simulation by including additional nodes that simulate edges and keep the edge state. This simulation seems more realistic as it allows to consider, as part of the complexity of the protocols, the increase in space due to maintaining information on every edge. Although nodes are not allowed to keep it as part of their internal state, which might require non constant space per node. In the following we define a simulation of a population protocol by a sensor field with additional nodes, those nodes represent the edges (links) in the communication graph. Observe that although in the communication graph directed arcs are possible, the way in which the transition function of the mediated population protocols is defined, makes that information is updated in both directions, independently on whether the edge is directed or undirected. To keep this property we add an additional node per link, this new node is connected through bidirectional links to both of its endpoints. As before, we define define the states and the transition function of the sensor field, then the mapping of input and scheduler to a tuple of input data streams. Observe that in mediated population protocols the nodes and the edge have distinguished roles. In our simulation we will assume that the first input on a data stream indicates the role of the node during the computation, and that the assignment is correct. Alternatively, we can assume that all the nodes are initially in state q_v and all the edges are initially in state q_e.

Given a MPP $\mathcal{P} = (X, Y, Q, S, I, O, \delta)$ running on interaction graph $G = (N, E)$, we define a CMSF $f(P, G) = (N', E', U, V, W, Q', \delta', q_0, q_v, q_e)$ as follows:

- The communication graph is the graph (N', E') where $N' = N \cup E$. The edge set is defined as follows

$$E' = \{(u, v) \mid u, v \in N and (u, v) \in E\} \cup \{((u, v), u)((u, v), v) \mid (u, v) \in E\}.$$

- $U = X \cup Y \cup \{a, r, e, \perp\}$
- $V = Y$ and $W = \{Q \cup S \cup \{\perp\}\}$
- $Q' = \{q_0, q_v, q_e\} \cup Q \cup S \cup (Q \times \{a, r\}) \cup (Q \times S \times \{a, r, e, w\}) \cup (Q \times Q \times S \cup S \times \{w, e\})$

The transition function δ' is again degined to keep track of the different communication rounds required in an interaction. In this case a total of 5 communications steps are required. The sending phases are depicted in Figure 5 and involve the three nodes participating in an interaction.

The input data stream $u(x, S)$ is the following:

- For any node $k \in V$ $u_k[1] = a$, for any node edge $(i, j) \in E$ $u_{(i,j)}[1] = e$.

- For any node k, $u_k[2] = x_k$. For any edge $u_{(i,j)}[2] = s_0$, were s_0 is the initial state.
- If the t-th interaction of the scheduler is (i, j), then
  - $u_i[5t-2, 5t+2] = aaaaa$
  - $u_j[5t-2, 5t+2] = rrrrr$
  - $u_{(i,j)}[5t-2, 5t+2] = eeeee$
  - $u_k[5t-2, 5t+2] = \perp\perp\perp\perp\perp$, for any $k \neq i, j, (i, j)$.

Theorem 11 ([49]). *Given a mediated population protocol $\mathcal{P}$ consider the CMSF $f(\mathcal{P})$ defined above. For any interaction graph G, mediated scheduler S, and input x consider the data stream $u(G, S, x)$ defined above. The t-th configuration of $\mathcal{P}$ on input x and scheduler S is identical to the $5t-2$-th configuration of $f(\mathcal{P})$ on input data stream $u(G, x, S)$.*

6. Deciding graph Languages

In [50] and in its more recent full version [51], the authors went one step further and studied what graph properties are stably computable by the MPP model. To understand properties of the interaction or communication graph is an important step in almost any distributed system. In particular, they temporarily disregarded the input notion and made the assumption that all agents simply start from a common *initial state* q_0. Also, as in the SMPP model, they made a similar assumption for the edges, that is, $\iota(e) = s_0$ for all $e \in E$ (recall that ι denotes the edge initialization function). Here the interest is in protocols that when executed on any communication graph G of a given graph universe, after a finite number of steps stabilize to configurations where all agents give 1 as output if G belongs to a graph language L, and 0 otherwise. This is motivated by the idea of having protocols that eventually accept all communication graphs (on which they run) that satisfy a specific property, and reject all remaining graphs.

A graph universe (family) $\mathcal{U}$ is closed under disjoint union if for any pair of graphs $G_1, G_2 \in \mathcal{U}$, the graph G obtained as the disjoint union of G_1 and G_2 belongs to $\mathcal{U}$.We consider the following graph families:

- $\mathcal{W}$ formed by all weakly connected digraphs without loops, and multiple edges.
- $\mathcal{S}$ formed by all strongly connected digraphs without loops, and multiple edges.
- $\mathcal{H}$ formed by all the directed graphs without isolated vertices, loops, and multiple edges.
- $\mathcal{G}$ formed by all possible directed graphs of any finite number of nodes greater or equal to 2.

To be able to perform some computation/communication, we have made the natural assumption, for the general case, that there are no isolated vertices on the communication graphs.

All the following definitions hold w.r.t. some fixed graph universe $\mathcal{U}$. A *graph language* L is a subset of $\mathcal{U}$ containing communication graphs that possibly share some common property. Some examples of graph languages are:

- The graph language consisting of all strongly connected members of $\mathcal{U}$.
- $L = \{G \in \mathcal{U} \mid G$ contains a directed hamiltonian path$\}$.
- $L = \{G \in \mathcal{U} \mid G$ has an even number of edges$\}$.
- $L = \{G \in \mathcal{U} \mid |V(G)| = |E(G)|\}$.

A graph language is said to be *trivial* if $L = \emptyset$ or $L = \mathcal{U}$.

A protocol $\mathcal{A}$ in model $\mathcal{M}$ is said to *stably decide* a graph language $L \subseteq \mathcal{U}$ (or equivalently a predicate $p_L : \mathcal{U} \to \{0, 1\}$ defined as $p_L(G) = 1$ iff $G \in L$) if for any $G \in \mathcal{U}$ and any computation of $\mathcal{A}$ on G, all agents eventually output 1 if $G \in L$ and all agents eventually output 0 if $G \notin L$. A graph language is said to be *stably decidable* by model $\mathcal{M}$ (also called $\mathcal{M}$-*decidable*) if some protocol $\mathcal{A}$ of $\mathcal{M}$ stably decides it.

A protocol $\mathcal{A}$ in model $\mathcal{M}$ is said to *stably recognize* a graph language $L \subseteq \mathcal{U}$ (or equivalently a predicate $p_L : \mathcal{U} \to \{0, 1\}$ defined as $p_L(G) = 1$ iff $G \in L$) if for any $G \in \mathcal{U}$ and any computation of $\mathcal{A}$ on G, all agents eventually output 1 if $G \in L$ and at least one agent outputs 0 if $G \notin L$. A graph language is said to be *stably recognizable* by model $\mathcal{M}$ (also called $\mathcal{M}$-*recognizable*) if some protocol $\mathcal{A}$ of $\mathcal{M}$ stably recognizes it.

6.1. Deciding graph Languages in the MPP model

To formally summarize, here the output alphabet is always binary, that is, $Y = \{0, 1\}$, $\iota(e) = s_0$ for all $e \in E$, and $I(\sigma) = q_0$, for all $\sigma \in X$ (due to this, an input alphabet is not specified neither is an input function and the assumption is that the initial configuration is $C_0(u) = q_0$ for all $u \in V$). This special case is called *Graph Decision Mediated Population Protocol* (GDMPP) model.

Theorem 12 ([51]). *The class of stably decidable graph languages by the GDMPP model is closed under complement, union, and intersection operations.*

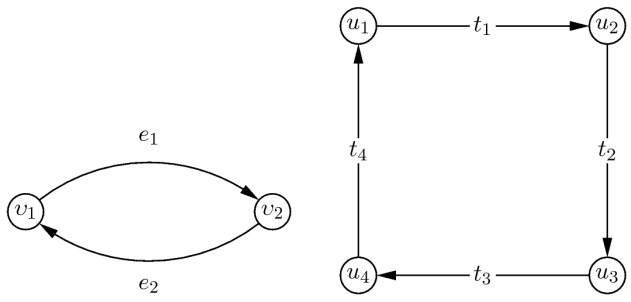
Node and edge parity, bounded out-degree by a constant, existence of a node with more incoming than outgoing neighbors, and existence of some directed path of length at least $k = \mathcal{O}(1)$ are some examples of stably decidable graph languages by the GDMPP model, in the case where the graph universe is $\mathcal{W}$. Also, given the same graph universe, the following impossibility result holds.

Theorem 13 ([51]). *There exists no GDMPP with stabilizing states to stably decide the graph language*

$$2C = \{G \in \mathcal{W} \mid \exists u, v \in V(G) \mid (u, v), (v, u) \in E(G)\}$$

(in words, G has at least one 2-cycle).

Proof Idea. We give the main idea of the proof with the help of Figure 6. Note that graph G (Figure 6(a)) has a 2-cycle, thus, belongs to $2C$, while G' (Figure 6(b)) does not. Assume that a GDMPP with stabilizing states stably decides $2C$ and take any computation of the protocol on G. Eventually, in such a computation, both agents output 1. Now, apply the same computation on G' in the following manner. Whenever e_1 is chosen in G choose first t_1 and then t_3 in G', and whenever e_2 is chosen in G choose t_2 and then t_4 in G'. It is not so hard to see that, due to symmetry, when the protocol state-stabilizes on G it also state-stabilizes with precisely the same states on G' (thus, also with the same output). Consequently, the protocol also accepts G' which is a contradiction and we reach the conclusion that no such protocol can exist. □



(a) Graph G (b) Graph G'

Figure 6: $G \in 2C$ and $G' \notin 2C$.

In the case where the graph universe is $\mathcal{G}$, containing all possible directed communication graphs (i.e. also the disconnected ones), it is possible to obtain a very strong impossibility result.

Lemma 6 ([51]). *For any non-trivial graph language L (L is* non-trivial *if $L \neq \emptyset$ and $L \neq \mathcal{G}$), there exists some disconnected graph G in L where at least one component of G does not belong to L or there exists some disconnected graph G' in $\overline{L}$ where at least one component of G' does not belong to $\overline{L}$ (or both).*

Theorem 14 ([51]). *Any non-trivial graph language $L \subset \mathcal{G}$ is not stably decidable by the GDMPP model.*

Proof Idea. The proof of the result is based on the very simple observation that in disconnected graphs, the various components cannot communicate with each other. Then Lemma 6 can be used to argue that a language (or its complement) must contain at least one disconnected graph with a component not in the language, so any protocol making some decision on the whole graph would make the opposite decision on the component (since this component does not belong to the language and is isolated from the other components), which is contradictory. □

Now, since the connectivity property is a non-trivial property, the following corollary comes as an immediate application of Theorem 14.

Corollary 2 ([51]). *The graph language*

$$C = \{G \in \mathcal{G} \mid G \text{ is (weakly) connected}\}$$

is not stably decidable by the GDMPP model.

6.2. Graph languages decided by constant memory sensor fields

Following the proposal in [51] the decidability of graph languages in the CMSF model has been analyzed in [49]. When the sensor field does not have access to any input data stream the set of decidable languages is too small. To have more significative results we allow only access to a particular input data stream. The input data stream will contain the information required to break the node symmetry, this information can be used to drive the computation in different ways. In this section we survey the results on graph language decidability for three variations of the CMSF model. The aim of the submodels is to explore different ways in which the input data stream can be used to guide the computation of a sensor field and compare the computational power of the resulting models by ways of the sets of graph languages that are decidable in the model.

6.2.1. The simplistic CMSF

This basic submodel assumes that we have a CMSF in which all the nodes have the same initial state and no input data stream is accessible. We call this model the simplistic CMSF.

In the case that we consider digraphs in a class that is closed under disjoint union, we have the same undecidability as in [51] concerning non-trivial properties. Observe that in this case, as communication between disconnected components cannot be established, a communication graph formed by several connected components in which there are components with positive answer and components with a negative answer cannot agree on a common answer.

Theorem 15 ([49]). *Let $\mathcal{U}$ be a graph family closed under disjoint union, then any non-trivial graph language in $\mathcal{U}$ is undecidable by simplistic CMSF.*

When the universe is the set of all strongly connected graphs non-trivial graph languages are undecidable. The proof is based on the fact that all the devices receive and send messages, because the graph is strongly connected. In such a case as all of them have the same initial state the computation is equivalent to a local computation in which the sent messages are the input messages for the following steps. Thus any decidable property must be independent of the communication graph.

Theorem 16 ([49]). *Any non-trivial graph language in $\mathcal{S}$ is undecidable by simplistic CMSF with topology in $\mathcal{S}$.*

When the graph universe is restricted to be $\mathcal{W}$ it is easy to find constructions in which information can be computed but not forwarded to all the nodes. Using this arguments it has been shown the following generic result.

Theorem 17 ([49]). *Any non-trivial graph property in $\mathcal{S}$ is undecidable by simplistic CMSF on $\mathcal{W}$.*

However in the simplistic CMSF model there are non-trivial decidable and undecidable properties when the communication graph belongs to $\mathcal{W}$. In decidable languages node symmetry is broken by some property that can be checked locally. Furthermore, the distinguished nodes must be able to broadcast this information to the rest of the network. The undecidable property obey to properties that can not be checked locally.

Theorem 18 ([49]). *In the simplistic CMSF, the graph language*

$$\{G \in \mathcal{W} \mid G \text{ has a node with no ingoing edges}\}$$

is decidable but the graph language

$$\{G \in \mathcal{W} \mid G \text{ has a node with no outgoing edges}\}$$

is undecidable

6.2.2. Node driven CMSF computation

The computational power of the simplistic CMSF has been enhanced by adding an input data stream that at any time step selects a node that can act as a leader for a local computation. This is equivalent to the scheduler in population protocols, but now the environment is selecting a node rather than an edge. On a valid computation it is required in addition fairness for the input data stream.

For most of the results a extension of the notion of fairness is required. A node scheduler S is an infinite sequence of nodes $v_1, v_2, \ldots$. For a constant k we say that a node scheduler is k-fair if the subsequence $(v_i)_{i \mod k=1}$ is fair. Therefore, we require that all the nodes appear infinitely often at the beginning of some periods of fixed length. All the definitions of decidability for this model will require stabilization on k-fair scheduler, for some constant k, that depends on the protocol. Observe that any fair scheduler can be padded to devise a k-fair scheduler, but that a fair scheduler might not be k-fair.

Theorem 19 ([49]). *Let $\mathcal{U}$ be a graph family closed under disjoint union, then any non-trivial graph language in $\mathcal{U}$ is undecidable by node driven CMSF.*

When the graph universe is restricted to be $\mathcal{W}$ we have the following generic results.

Theorem 20 ([49]). *Any non-trivial graph property in $\mathcal{S}$ is undecidable by node driven CMSF on $\mathcal{W}$.*

But not taking into account the input data stream we have the following result.

Theorem 21 ([49]). *Any graph property decidable by simplistic CMSF on $\mathcal{W}$ is decidable by node driven CMSF on $\mathcal{W}$.*

However when restricted to strongly connected graphs some non-trivial graph languages become decidable in the node driven model. The main idea is to use a controlled broadcast, consisting of a fixed number of steps, that allows to check the property on the local neighborhood of the activated node. This results on a scheduler that stabilizes on any k-fair scheduler for some adequate constant k that depends on the designed protocol.

Theorem 22 ([49]). *The following properties are decidable by node driven CMSF with communication topology in $\mathcal{S}$.*

- *$kC = \{G \in \mathcal{S} \mid G$ contains a k-cycle$\}$, for any constant k.*
- *Directed simple path of constant length.*

Observe that kC is a non-trivial graph property (even on strongly and weakly connected graphs) and therefore the language is not decidable in the simplistic CMSF neither in the node driven CMSF on $\mathcal{W}$.

6.2.3. Edge driven CMSF computation

Another extension is to consider the CMSF that has been used to simulate the computation of a mediated population protocol. Here the communication graph has additional nodes that keep the state of the edge. In addition it is assumed that all the nodes corresponding to vertices in the communication graph have the same initial state and all the nodes corresponding to edges also have the same initial state. However, the initial state of edges can be different to the initial state of vertices. The data stream will select one edge per time step, thus activating the three participants on the interaction, the actuator, the respondent, and the edge. On a valid computation we assume in addition k-fairness, for some fixed constant k.

This computational model has more computational power than the GDMPP model. The result follows from using a variation of the simulation MPP in which the repetition of the values on the data stream is replaced by local states in order to guarantee that the protocol stabilizes on 5-fair edge schedulers.

Theorem 23 ([49]). *Any graph property that is decidable by GDMPP is decidable by edge driven CMSF.*

Furthermore, the edge driven CMSF has also more computational power than the node driven CMSF model. Here in any activation of an edge (u, v), the simulation simulates first the steps taken by a node scheduler that activates u and continues simulating the activation of v. This leads to the construction of a k-fair node scheduler from a $2k$-fair edge scheduler.

Theorem 24 ([49]). *Any graph property that is decidable by node driven CMSF is decidable by edge driven CMSF.*

7. Conclusions - Open Problems

We surveyed some very recently proposed computational models for networks of tiny artifacts. For the case of interaction based models we focused on the area of population protocols. We discussed some recent enhancements, namely, the MPP model and the PM model. Both models are the result of adding some extra feature to the population protocol model. The former assumes that the communication links can store limited information and the latter that each agent is a multitape TM. We discussed some very recent research proving that both models are extremely powerful in terms of their computational power. In complete graphs, the MPP model and the PM model for every $f(n) = \Omega(\log n)$ space bound are equivalent to nondeterministic TMs of $\mathcal{O}(n^2)$ and $\mathcal{O}(nf(n))$ space, respectively, that compute symmetric predicates. In particular, we presented *Static Synchronous Sensor Field* (SSSF) which consists of a *set of devices* and a *communication graph*. The computation of a SSSF depends on the communication between the devices and the environment. We reviewed the computational problems that are susceptible of being solved by sensor fields and we analyzed the computational resources used to solve them. We discussed several memory restrictions of the *tiny* devices involved in a SSSF allowing only devices with constant or bounded memory capacity. We analyzed the computational power of sensor fields in the particular case in which the memory per device is constant, the CMSF model, in relation with the Population Protocol models, showing adequate simulations of population protocols by sensor fields. We also analyzed the decidability of properties of the communication or interaction graph under different variants of PP and CMSF.

Many interesting problems remain open in the area of population protocols:

- Are the MPP and PM models fault-tolerant? What preconditions are needed in order to achieve satisfactory fault-tolerance?
- In order to apply our protocols in real and critical application scenarios some sort of code verification seems necessary. A first attempt for the basic population protocol model can be found in [52]. Verification methods for MPPs, Community Protocols, and PM protocols are still totally unknown, although the ideas of [52] may also be applicable to these models.
- [53] revealed the need for population protocols to have adaptation capabilities in order to keep working correctly and/or fast when natural modifications of the mobility pattern occur. However, we do not know yet how to achieve *adaptivity*.
- Is there an exact characterization of the class of stably decidable graph languages by GDMPP in the weakly-connected case?
- $\mathcal{O}(\log n)$ memory per agent seems to behave as a *threshold*. Is there some sort of impossibility result showing that with $\mathcal{O}(f(n))$ memory, where $f(n)$ is asymptotically smaller than $\log n$, the class of stably computable predicates is strictly smaller than $SNSPACE(nf(n))$? At a first glance, it seems that the agents are unable to store uids and get informed of the population size.
- Study the stable decidability of graph languages by the PM model for various space bounds (the most prominent being the logarithmic bound).

In the area of sensor fields, among many others, we can post the following open directions:

- Is there a characterization of the sensing problems that can be solved in logarithmic or constant space?
- Allow the incorporation of a *dynamic communication graph*. This requires a slight modifcation of the definition of the sensor field, just allowing that the sets of neighbors change dynamically at any time step. The dynamicity in the communication graph can come from faults (either nodes or links) or from mobility, or both. Finding the paradigmatic family of problems for such networks and the adequeate complexity measures will require an additional modeling effort.
- Consider and incorporate energy models to the communication graph. For analyzing the energy consumption the performance measures proposed so far proportionate the basic ingredients for the case that sending/receiving a message has uniform cost for all the nodes. Further basic complexity measures can be required for more complex energy models. The aim should be in the design of optimal (efficient) algorithms under different energy models.
- In the CMSF model all the results are obtained for the arbitrary model of communication. To some extent most of the results can be extended to the priority model. A good understanding of the collision model will be of interest as losing information due to collisions is a problem that naturally arises in wireless networks.

References

[1] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, R. Peralta, Computation in networks of passively mobile finite-state sensors, in: PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing, ACM, New York, NY, USA, 2004, pp. 290–299.

[2] D. Estrin, R. Govindan, J. Heidemann, S. Kumar, Next century challenges: Scalable coordination in sensor networks, in: ACM/IEEE Int. Conf. on Mobile Computing and Networking, Seattle, pp. 263–270.

[3] B. Warneke, M. Last, B. Liebowitz, K. Pister, Smart dust: Communicating with a cubic-millimeter computer, Computer 34 (2001) 44–51.

[4] J. M. Kahn, R. H. Katz, K. S. J. Pister, Emerging challenges: Mobile networking for smart dust, Journal of Communications and Networks 2 (2000) 188–196.

[5] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A Survey on Sensor Networks, IEEE Comunnications Magazine 40 (2002) 102–114.

[6] D. Estrin, D. Culler, K. Pister, G. Sukatme, Connecting the Physical World with Pervasive Networks, IEEE Pervasive Computing 6 (2002) 59– 69.

[7] S. Tilak, N. Abu-Ghazaleh, W. Heinzelman, A Taxonomy of Wireless Micro-Sensor Network Models, Mobile Computing and Communications Review 6 (2003) 28–36.

[8] M. Vinyals, J. Rodriguez-Aguilar, J. Cerquides, A Survey on Sensor Networks from a Multi-Agent Perspective, in: Proc. of AAMAS 2008.

[9] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, R. Peralta, Computation in networks of passively mobile finite-state sensors, Distributed Computing (2006) 235–253.

[10] I. Chatzigiannakis, O. Michail, P. G. Spirakis, Mediated population protocols, in: 36th International Colloquium on Automata, Languages and Programming (ICALP 2009). Part 2, volume 5556 of *LNCS*, pp. 363–374.

[11] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, P. G. Spirakis, Passively mobile communicating logarithmic space machines, Technical Report FRONTS-TR-2010-16, RACTI, Patras, Greece, 2010.

[12] D. T. Gillespie, Exact stochastic simulation of coupled chemical reactions, The Journal of Physical Chemistry 81 (1977) 2340–2361.

[13] D. T. Gillespie, A rigorous derivation of the chemical master equation, Physica A 188 (1992) 404–425.

[14] J. Aspnes, E. Ruppert, An introduction to population protocols, Bulletin of the European Association for Theoretical Computer Science 93 (2007) 98–117.

[15] Z. Diamadi, M. J. Fischer, A simple game for the study of trust in distributed systems, Wuhan University Journal of Natural Sciences 6 (2001) 72–82. Also appears as Yale Technical Report TR1207, Jan. 2001.

[16] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, R. Peralta, Urn automata, Technical Report Technical Report YALEU/DCS/TR-1280, Yale University Department of Computer Science, 2003.

[17] T. G. Kurtz, Approximation of population processes, Number 36 in CBMS-NSF Regional Conference Series in Applied Mathematics (1981).

[18] J. Berstel, Quelques applications des reseaux d'automates, Thèse de 3ème Cycle, 1967.

[19] P. Gibbons, S. Tirthapura, Estimating simple functions on the union of data streams, in: SPAA, pp. 281–291.

[20] C. Hoare, A calculus of total correctness for communicating processes, Sci. Comput. Program. 1 (1981) 49–72.

[21] D. Peleg, Distributed Computing. A Locality-Sensitive Approach, SIAM Monographs on Discrete Mathematics and Applications.

[22] T. Henzinger, The Theory of Hybrid Automata, in: Proceedings of the 11th Annual IEEE Symposium (LICS 96), pp. 278–292.

[23] N. Lynch, R. Segala, F. Vaandrager, Hybrid I/O Automata Revisited, in: Hybrid Systems: Computation and Control: 4th International Workshop (HSCC'01), volume 2034 of *LNCS*, pp. 403–417.

[24] I. Chatzigiannakis, S. Nikoletseas, P. Spirakis, Smart dust local detection and propagation protocols, in: Proceedings of the second ACM international workshop on Principles of mobile computing, pp. 9–16.

[25] J. Díaz, J. Petit, M. Serna, A random graph model for optical smart dust networks, IEEE Transactions on Mobile Computing 2 (2003) 186–196.

[26] G. Arzhantseva, J. Díaz, J. Petit, J. Rolim, M. Serna, Broadcasting on networks of sensors communicating through directional antennas, in: P. Spirakis, A. Kameas, S. Nikoletseas (Eds.), International Workshop on Ambient Intelligence Computing, Ellinika Grammata, CTI Press, 2003, pp. 1–12.

[27] M. Penrose, Random Geometric Graphs, Oxford Studies in Probability, Oxford U.P., 2003.

[28] F. Xue, P. Kumar., The number of neighbors needed for connectivity of wireless networks, Wireless Networks 10 (2004) 169–181.

[29] F. T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann, San Mateo, CA, 1993.

[30] C. Àlvarez, A. Duch, J. Gabarro, M. Serna, Sensor field: A computational model, in: Algorithmic Aspects of Wireless Sensor Networks: 5th International Workshop, ALGOSENSORS 2009, Rhodes, Greece, July 10-11, 2009. Revised Selected Papers, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 3–14.

[31] L. Valiant, A bridging model for parallel computation, Commun. ACM 33 (1990) 103–111.

[32] J. von Neumann, Theory and organization of complicated automata, in: A. Burks (Ed.), Theory of Self-Reproducing Automata [by] John von Neumann, University of Illinois Press, Urbana (1949), 1949, pp. 29–87 (Part One). Based on transcripts of lectures delivered at the University of Illinois, in December 1949. Edited for publication by A.W. Burks.

[33] D. Angluin, J. Aspnes, D. Eisenstat, E. Ruppert, The computational power of population protocols, Distributed Computing 20 (2007) 279–304.

[34] R. Guerraoui, E. Ruppert, Names trump malice: Tiny mobile agents can tolerate byzantine failures, in: ICALP (2), pp. 484–495.

[35] S. Ginsburg, E. H. Spanier, Semigroups, presburger formulas, and languages, Pacific Journal of Mathematics 16 (1966) 285–296.

[36] M. Presburger, Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt, in: Comptes-Rendus du I Congrès de Mathématiciens des Pays Slaves, pp. 92–101.

[37] D. Angluin, J. Aspnes, D. Eisenstat, Stably computable predicates are semilinear, in: PODC '06: Proceedings of the 25th annual ACM Symposium on Principles of Distributed Computing, ACM Press, New York, NY, USA, 2006, pp. 292–299.

[38] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, E. Ruppert, When birds die: Making population protocols fault-tolerant, in: DCOSS, pp. 51–66.

[39] N. A. Lynch, Distributed Algorithms, Morgan Kaufmann; 1st edition, 1996.

[40] P. G. Spirakis, Population protocols and related models, in: S. Nikoletseas, J. Rolim (Eds.), Theoretical Aspects of Distributed Computing in Sensor Networks, Springer-Verlag, 2010. To appear.

[41] D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, R. Peralta, Stably computable properties of network graphs, in: V. K. Prasanna, S. Iyengar, P. Spirakis, M. Welsh (Eds.), Distributed Computing in Sensor Systems: First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USE, June/July, 2005, Proceedings, volume 3560 of *Lecture Notes in Computer Science*, Springer-Verlag, 2005, pp. 63–74.

[42] D. Angluin, J. Aspnes, D. Eisenstat, Fast computation by population protocols with a leader, Distributed Computing 21 (2008) 183–199.

[43] O. Bournez, P. Chassaing, J. Cohen, L. Gerin, X. Koegler, On the convergence of population protocols when population goes to infinity, Applied Mathematics and Computation (2009). To appear.

[44] I. Chatzigiannakis, P. G. Spirakis, The dynamics of probabilistic population protocols, in: DISC '08: Proceedings of the 22nd international symposium on Distributed Computing, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 498–499.

[45] O. Bournez, J. Chalopin, J. Cohen, X. Koegler, Playing with population protocols, in: CSP, pp. 3–15.

[46] I. Chatzigiannakis, O. Michail, P. G. Spirakis, Recent advances in population protocols, in: MFCS '09: Proceedings of the 34th International Symposium on Mathematical Foundations

of Computer Science 2009, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 56–76.
[47] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, P. G. Spirakis, All symmetric predicates in $NSPACE(n^2)$ are stably computable by the mediated population protocol model, in: MFCS '10: Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science 2010. To appear.
[48] M. Sipser, Introduction to the Theory of Computation, Second Edition, International Edition, Thomson Course Technology, 2006.
[49] C. Àlvarez, M. Serna, P. G. Spirakis, On the computational power of constant memory sensor fields, Technical Report FRONTS-TR-2010-10, 2010.
[50] I. Chatzigiannakis, O. Michail, P. G. Spirakis, Brief announcement: Decidable graph languages by mediated population protocols, in: DISC, pp. 239–240.
[51] I. Chatzigiannakis, O. Michail, P. G. Spirakis, Stably decidable graph languages by mediated population protocols, in: 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2010), LNCS. To appear.
[52] I. Chatzigiannakis, O. Michail, P. G. Spirakis, Algorithmic verification of population protocols, in: 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2010), LNCS. To appear.
[53] I. Chatzigiannakis, S. Dolev, S. P. Fekete, O. Michail, P. G. Spirakis, Not all fair probabilistic schedulers are equivalent, in: OPODIS '09: Proceedings of the 13th International Conference on Principles of Distributed Systems, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 33–47.

Prof. Carme Àlvarez obtained her Ph.D. from Technical University of Catalonia (UPC), in 1994. She is currently Professor in the UPC, Barcelona. Her research interests are Computational Complexity, Models of Computation, and Algorithmic Game Theory. Carme Àlvarez has published in important Computer Science journals and significant refereed conferences.

Dr. Ioannis Chatzigiannakis obtained his Ph.D. from the Department of Computer Engineering & Informatics of the University of Patras in 2003. He is currently Adjunct Faculty at the Computer Engineering & Informatics Department of the University of Patras (since October 2005). He is the Director of the Research Unit 1 of RACTI (since July 2007). He has coauthored over 70 scientific publications. His main research interests include distributed and mobile computing, wireless sensor networks, algorithm engineering and software systems. He has served as a consultant to major Greek computing industries. He is the Secretary of the European Association for Theoretical Computer Science since July 2008.

Dr. Amalia Duch obtained her PhD from the Department of Languages and Informatics Systems of the Technical University of Catalonia in 2004. She is currently an Associate Professor of the Department of Languages and Informatics Systems of the Technical University of Catalonia (since September 2005). Her research interests include Average-case Analysis of Algorithms, Multidimensional Data Structures and, recently, Algorithmic Game Theory.

Prof. Joaquim Gabarro, was born in Barcelona (Spain) in 1953 and studied Physics and Computer Science, obtained his Ph.D. from Universitat Politècnica de Calatunya in 1983. From 1985 is Associate Professor (Profesor Titular) in Universitat Politècnica de Calatunya. Currently represents Spain in the TC1 (Foundations of Computer Sciences) of IFIP. Main research topics are Concurrency, Algorithms and Complexity.

Othon Michail, born in 1984, obtained his Diploma and his MSc in Computer Science & Technology from the Department of Computer Engineering & Informatics of the University of Patras in 2007 and 2009, respectively. He is currently a Ph.D. student at the Computer Engineering & Informatics Department of the University of Patras (since June 2009) under the supervision of Prof. Dr. Paul Spirakis and he is also a member of the Research Unit 1 of the RACTI (since April 2009). His research interests include Theory of Computation in new models of computation (like wireless sensor networks and, in particular, population protocols), Computational Complexity, and Algorithms.

Prof. Dr. Maria Serna obtained her Ph.D. from Technical University of Catalonia (UPC) in 1990. She is currently Full Professor in the Department of Languages and Informatics Systems (LSI) of UPC. She is the coordinator of the Computing Postgraduate program at LSI. She has co-autored over 90 scientific publications. Her main research interest are algorithms and complexity, parallel and distributed computing, problems on graphs and networks, and algorithmic game theory. She is the editor of the Bulletin of the EATCS.

Prof. Dr. Paul Spirakis, born in 1955, obtained his Ph.D. from Harvard University, in 1982. He is currently the Director of the RACTI and a Full Professor in the Patras University, Greece. He was acknowledged between the top 50 scientists worldwide in Computer Science with respect to The best Nurturers in Computer Science Research, published by B. Kumar and Y.N. Srikant, ACM Data Mining, 2005. His research interests are Algorithms and Complexity and interaction of Complexity and Game Theory. Paul Spirakis has extensively published in most of the important Computer Science journals and most of the significant refereed conferences contributing to over 200 scientific publications. He was elected unanimously as one of the two Vice Presidents of the Council of the EATCS.