IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

"Semi-Asynchronous": a new scheduler in distributed computing

SERAFINO CICERONE[1], GABRIELE DI STEFANO[1], and ALFREDO NAVARRA.[2]
[1]Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Via Vetoio, I–67100, L'Aquila, Italy
[2]Department of Mathematics and Computer Science, University of Perugia, Via Vanvitelli 1, I–06123, Perugia, Italy.

Corresponding author: Alfredo Navarra (e-mail: alfredo.navarra@unipg.it).

A preliminary version of some of these results appeared in the Proceedings of the 38th IEEE International Conference on Distributed Computing Systems (ICDCS), 2018 [10].

ABSTRACT The study of mobile entities that based on local information have to accomplish global tasks is of main interest for the scientific community. Classic models for the activation and synchronization of mobile entities are the *fully-synchronous* (FSYNC), *semi-synchronous* (SSYNC), and *asynchronous* (ASYNC) models, where entities alternate between *active* and *inactive* states with different timing.
According to the assumed synchronization model, very different results have been achieved in the field of distributed computing. One of the main outcomes is the big gap between the ASYNC and the other models in terms of manageability and algorithm design. In fact, there are still many problems for which it is not known whether synchronicity is crucial for designing resolution algorithms or not.
In order to better understand the ASYNC case, here we propose a further model referred to as the *semi-asynchronous* (SASYNC). This slightly deviates from SSYNC. In fact, like in SSYNC (and FSYNC), the duration of the activation of an entity is kept of fixed time whereas, like in ASYNC, the starting instant of the activation is not fully synchronized with the possible activation of other entities.
We show that for entities moving on graphs, the SSYNC model allows accomplishing more tasks than the SASYNC that in turn allows accomplishing more tasks than the ASYNC.
Furthermore, our results show that, especially to tackle problems in the Euclidean plane, the SASYNC model is already quite challenging, therefore there is no need to get involved with complications arising in the ASYNC model.

INDEX TERMS Distributed Algorithms; Gathering; Mobile Robots; Synchronization.

I. INTRODUCTION

In this paper, we investigate on feasibility issues in distributed computing systems. We consider global tasks for autonomous entities endowed with very weak capabilities. As a standard notation, entities are usually referred to as *robots* in the literature and so we do in the rest of the paper.

Standard and basic assumptions about capabilities consider robots to be: *autonomous*, there is no central coordination for their actions; *disoriented*, each robot refers to its own local coordinate system that may have no relation with those of other robots; *anonymous*, they are indistinguishable and are not associated with any ids; *homogeneous*, they all have the same capabilities and execute the same (deterministic) algorithm; *dimensionless*, they are perceived as points in the Euclidean plane or as occupying nodes of a graph; *silent*, they have no direct means of communication, that is, although they can see to each other, they cannot explicitly communicate. It follows that all synchronizations, interactions, and information spreadings among the robots take place solely by observing the position of the robots in the moving environment.

Concerning the behavior of the robots, one of the most studied scenario assumes each robot to alternate between *active* and *inactive* periods. When active, a robot operates in standard *Look-Compute-Move* (*LCM*) computational cycles, see e.g. [8], [39], [52]. Within one cycle, a robot acquires a snapshot of the current global positioning of the other robots (Look phase) with respect to its own coordinate system. Successively, in the Compute phase, it decides whether to move toward a computed direction or not (i.e., it executes the designed distributed algorithm), and then performs the move (Move phase), possibly a *nil* movement.

When dealing with such weak distributed robot systems, it comes out that the feasibility of a global task depends on

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

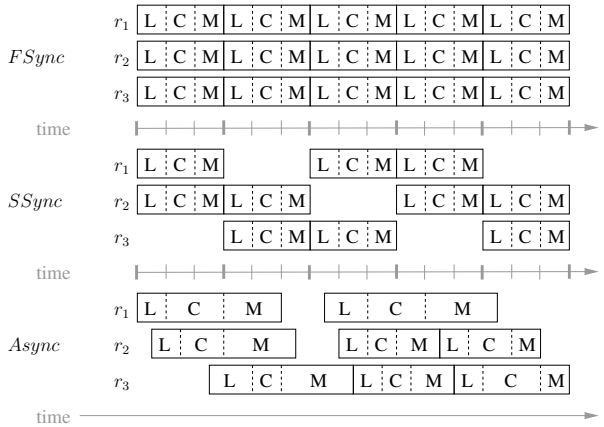


FIGURE 1: The Look-Compute-Move cycles that may occur during an execution referring to the FSYNC, SSYNC, or ASYNC models. For FSYNC and SSYNC, notice the subdivision of the time axis into time units and the mark related to each round.

many factors. The described robots' behavior and capabilities are certainly crucial but also the *magnitude of synchrony* among the robots represents a main issue.

To this respect, in the literature, different characterizations of the environment, illustrated in Figure 1, have been considered according whether robots are fully-synchronous, semi-synchronous (cf. [14], [15], [52]–[54]) or asynchronous (cf. [1], [24], [38], [41], [42]):

- *Fully-Synchronous* (FSYNC): All robots are always active, continuously executing their LCM cycles in a synchronized way. Hence, the time can be logically divided into global rounds. In each round all the robots, obtain a snapshot of the environment, compute on the basis of the obtained snapshot and perform their computed move.
- *Semi-Synchronous* (SSYNC): It is basically equivalent to the FSYNC model, with the only difference that within a round not all robots must be active.
- *Asynchronous* (ASYNC): Robots are activated independently. Moreover, each phase may last for an undefined but finite time. As a result, robots do not have a common notion of time. Moreover, they can be seen while moving, and computations can be made based on obsolete information about positions.

In ASYNC, according to [38], [51], it is possible to assume without loss of generality that the snapshot obtained during the Look phase is in fact acquired at the beginning of such a phase. The rationale behind it is that the Look phase can be potentially thought as composed of three sub-phases: (i) activation of the sensors; (ii) instantaneous snapshot acquisition; (iii) processing data. Hence, by considering sub-phase (i) as part of the preceding inactivity phase, the assumption stands. This clearly does not change the computational power of ASYNC but it reveals to be very useful when the behavior of the robots is analyzed. We then assume such a constraint in all the synchronization models.

Furthermore, it is usually assumed that for SSYNC and ASYNC schedulers, an ideal *adversary* determines the Look-Compute-Move cycles timing (i.e., which robots are activated at a given time). Anyway, the scheduler is always assumed to be *wait-free*, that is each robot is activated, eventually, and within finite time. Hence within a finite but unpredictable window of time, each robot executes its LCM cycle. This assumption is necessary to guarantee the evolution of the system as, otherwise, an adversary may take it unchanged forever.

One crucial property of the ASYNC model, that does not apply in FSYNC nor in SSYNC, concerns the possibility for the robots to be perceived while they are moving or while robots have decided to move but they have not yet started moving. In fact, as shown in Figure 1, during the Look phase of a robot in ASYNC, other robots can be in any other phase, whereas this cannot happen in FSYNC and SSYNC due to the synchronization dictated by such models. It follows that, in ASYNC a move can be performed after a long time from the moment it has been computed, that is the current configuration might have drastically changed with respect to that moment. Such occurrences might heavily affect the study of the ASYNC model where it might be very difficult sometimes to figure out what is going on. Hence, ASYNC reveals to be a model much more hostile with respect to FSYNC and SSYNC for designing distributed algorithms. This, of course, extraordinarily affects also the arguments necessary to provide the proofs of correctness of the proposed algorithms. To this respect, see [11] for an extended discussion related to the difficulties encountered when dealing with ASYNC.

In this paper, our investigation is toward the definition of a new synchronization model that may exploit some synchronization issues, like in FSYNC and SSYNC, but still preserves some properties specific of the ASYNC model. In practice, we look for a model that slightly deviates from SSYNC toward ASYNC. Our aim is to better catch the peculiarities that make ASYNC harder than SSYNC to deal with, but without incurring in all the difficulties specific of ASYNC. We will call our new model as *semy-asynchronous*, SASYNC in brief. Such a model will help to better understand the big gap between SSYNC and ASYNC, in terms of computational power. To this respect, our investigation heavily involves the study of the *Gathering* problem. In general, this is the requirement, for a given set of robots, to let them meet, eventually.

A. OUTLINE

The paper is organized as follows. In the next section, we revise some related literature. In Section III, we formally define our new synchronization model. Namely, we introduce the so-called SASYNC model, which lies in between SSYNC and ASYNC in terms of computational power. Successively, we start discussing about its properties and then describe an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

overview of our achievements. In Section IV, we formally define the *Gathering* problem. In Section V, we provide results for the studied Gathering problem with respect to SSYNC robots moving on a ring, while the same problem is studied for SASYNC robots in Section VI. In Section VII, we analyze the new model with respect to the Gathering problem for robots moving on the Euclidean plane. Finally, in Section VIII, we conclude the paper by highlighting some challenging research directions.

II. RELATED WORK

The research, in the field described in Introduction, has mainly studied feasibility issues. In particular, for the most of the tasks, the question has been whether such tasks are solvable or if further assumptions must be introduced. The choice of the synchronization model very often determines whether a task is feasible or not. Tasks well investigated to this respect are the *Gathering*, see [5]–[7], [9], [18], [21], [22], [31], [32], [37], [43], [44], [46], in which all robots are required to reach a common destination not known in advance; the *Pattern formation*, see [4], [11], [33], [40], [41], in which robots are required to form a specific geometric pattern in the Euclidean plane or to suitable dispose on the nodes of a graph; the *Leader Election* problem, see [8], [19], [20], [33], where one robot (when possible) must be selected and recognized by all the others as the leader; the *Exploration* problem, see [3], [17], [25], [27], [29], [34]–[36], [47], [48], where the robots are required to visit / explore an area of interest or a graph.

The environment where robots move can be the Euclidean space [35], [54] or a graph where robots are constrained to follow the path dictated by the edges [12], [13], [17]–[19], [27]. Robots might be endowed with communication means, e.g. by means of tokens as in [34], or they only exploit stigmergic properties, see e.g. [36]. Along with feasibility, sometimes also optimization issues have been explored, see [5], [7], [9], [31], [32]. Objective functions to accomplish a specific task may refer to the number of robots as in [27], or the number of LCM cycles as in [15].

Our work has been mainly inspired by [23], [26], [28], [51]. Such papers concern how synchronization impacts on the resolution of problems in the LCM context.

As defined in [26], here we use the same notation to compare synchronization models with respect to the induced computational power. In particular, given two synchronization models $\mathcal{M}$ and $\mathcal{N}$, inequality $\mathcal{M} \geq \mathcal{N}$ means that any task that can be solved in $\mathcal{N}$ is also solvable in $\mathcal{M}$. In other words, $\mathcal{M}$ is not less powerful than $\mathcal{N}$. Inequality $\mathcal{M} > \mathcal{N}$ means that $\mathcal{M} \geq \mathcal{N}$ holds and there exists a task that can be solved in $\mathcal{M}$ but not in $\mathcal{N}$. In other words, $\mathcal{M}$ is more powerful than $\mathcal{N}$.

As described in [23], it is known that FSYNC $>$ SSYNC. The inequality has been obtained by showing that the *Rendezvous* problem (that is, the Gathering of two robots), in the Euclidian plane, is solvable by means of FSYNC robots but not by SSYNC robots (see, e.g. [52]).

Another interesting result about synchronization model is that SSYNC $>$ ASYNC. This inequality has been obtained by introducing the so-called *Movement Awareness* problem in the Euclidian plane that is solvable by means of robots equipped with some memory in the SSYNC model but not in the ASYNC model (see, e.g. [51]).

III. THE SASYNC MODEL

In this section, we first define the new SASYNC synchronization model and provide some useful outcoming properties. Then, we discuss about the roadmap of our investigation on SASYNC.

A. DEFINING SASYNC

We now provide the formal definition of our new synchronization model:

- *Semi-Asynchronous* (SASYNC): Similarly to ASYNC, robots are activated independently so that two robots can be in different phases. Like in FSYNC or SSYNC, robots are synchronized with respect to phases, that is two robots in two different phases started such phases concurrently. Moreover, the duration of an LCM cycle equals exactly two time units, the first one associated with the Look (instantaneously executed at the beginning) and the Compute phases, whereas the second one to the Move phase (cf. Figure 2).

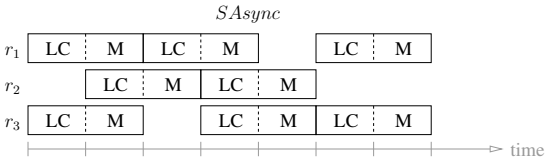


FIGURE 2: The Look-Compute-Move cycles that may occur during an execution referring to the SASYNC model.

Clearly, the wait-freedom of the adversary is maintained in the SASYNC model as well as the assumption by which the snapshot obtained during the Look phase is acquired at the beginning of such a phase. Moreover, in SASYNC a robot cannot infer from the snapshot whether other robots are inactive or performing one of the phases of the LCM cycle. Hence, similarly to ASYNC, robots can move based on a configuration that has changed meanwhile.

Remark III.1. *According to the defined* SASYNC *model, the order in which the moves are performed equals exactly the order in which robots have been activated (FIFO behavior). This represents the main difference with respect to* ASYNC *where instead moves can be performed in any order. In particular, in* ASYNC *a move computed later can occur before moves computed earlier. In* SASYNC*, instead, each LCM cycle lasts for exactly the same amount of time for each robot. Hence, although the Look phase performed by a robot r might overlap the Move phase performed by any other robot r', the move of r' will be effective always before the move of r.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

Definition III.2. *A move that has been computed from a configuration that is not the current one is called pending if it has not been performed yet.*

It turns out that, in ASYNC, there might occur a pending move that has been computed very far away in time, whereas in SASYNC a pending move can refer only to *at most one time unit* back in time (where a time unit refers to one half of the duration of the whole LCM cycle, cf. Figure 2).

When robots move within the Euclidean space, movements can be assumed of two types: a movement is *rigid* when a robot is always ensured to reach its target point within one LCM cycle; a movement is *non-rigid* if the robot does not have such a guarantee, that is the adversary can stop it before reaching the target. However, within one move the length of the trajectory traced is neither infinitesimally small nor infinite. In particular, the adversary may prevent a robot to reach its destination as long as such a point is further than a distance $\delta > 0$, unknown to the robots. In such a case, the robot is only ensured to move of at least δ. Whenever the destination is within distance δ, then the robot is ensured to reach it. Clearly, without introducing the assumption on the constant δ, the adversary may prevent a robot to ever reach its destination. When robots move along the edges of a graphs, a move is realized by reaching a neighboring node as destination, and it is assumed to be instantaneous, hence rigid. It follows that robots are never perceived on the edges but always on the nodes.

Remark III.3. *Differently from* ASYNC*, if a robot r in* SASYNC *is performing the Move phase, while another robot r' is performing the Look phase, then r' cannot perceive the move of r. In fact, r' acquires the snapshot at the beginning of the LCM cycle, that is before r starts its movement. It follows that robots cannot be seen while they are moving. However, differently from* SSYNC*, robots in* SASYNC *can be seen while they have already decided where to move.*

Remark III.4. *The introduced* SASYNC *model could be easily extended by defining further levels of obsolescence for the pending moves. Starting by observing that, in the introduced* SASYNC *model, a pending move cannot stand as such for more than one time unit (whereas in* ASYNC *a pending move can last for a finite but unpredictable time), we could define a hierarchy of models by using an integer parameter $k \geq 1$ to represent the number of time units a pending move can stand. In any of such models, the FIFO behavior would be still preserved because of the fixed dimension of the LCM cycle of $k+1$ time units. However, for our purposes we prefer to deal with just the basic case of the defined* SASYNC*, hence deviating from* SSYNC *as less as possible.*

B. INVESTIGATION OVERVIEW

Our aim is to find out the relations holding among the various synchronization models, including SASYNC, in terms of computable tasks. The next theorem anticipates one of the main results achieved within this paper. It relates the computational power of SASYNC with that of the SSYNC and ASYNC models for robots moving on graphs.

Theorem III.5. *For robots moving on graphs, it holds:* SSYNC $>$ SASYNC $>$ ASYNC.

By definition, it trivially holds SSYNC $\geq$ SASYNC $\geq$ ASYNC also for robots moving in the Euclidean plane. However, Theorem III.5 shows that the strict inequalities hold. In particular, on graphs the SSYNC model is computationally more powerful (i.e. more tasks can be solved) than the SASYNC model, that in turn is computationally more powerful than the ASYNC model.

In order to prove Theorem III.5, we have deeply investigated, with respect to the various synchronization models, the Gathering problem for robots moving on a ring. The Gathering problem on rings has been almost completely characterized in [21] for ASYNC robots. In particular, there are some configurations that have been classified as *unsolvable*, i.e. configurations from which the Gathering problem cannot be solved. For robots moving on rings, unsolvable configurations are those with exactly two robots and those admitting some specific symmetries. For all the other ones, but the class of the so-called $\mathcal{SP}4$ configurations, a resolution algorithm has been provided. A configuration is of type $\mathcal{SP}4$ if it admits a geometrical axis of reflection, with four robots on a ring with an odd number of nodes and some constraints on the disposal of the robots (the formal definition of $\mathcal{SP}4$ configurations will be provided later). Hence, whether $\mathcal{SP}4$ configurations are solvable or not in ASYNC remains an open problem.

Theorem III.5 is obtained by specifically studying the Gathering problem on configurations in $\mathcal{SP}4$. In particular, we show that SSYNC robots suffice to always solve the Gathering from configurations of four robots on rings with $n \geq 7$ nodes. Considering instead robots in SASYNC, we provide an algorithm to solve the case of four robots on seven-node rings, whereas we prove that $\mathcal{SP}4$ configurations with $n = 9$ nodes are unsolvable. Since in [30] it has been proven that the case of four robots on a seven-node ring in ASYNC is unsolvable, then Theorem III.5 follows.

Interestingly, our investigation also leads to fully characterize the Gathering problem for robots moving on rings in the SSYNC model.

Finally, we provide some evidence about the relevance of the SASYNC model in the context of robots moving in the Euclidean plane. In particular, we show that the resolution of the Gathering problem of SASYNC robots seems to maintain the same difficulties encountered in the ASYNC context, hence suggesting that SASYNC might be a "sufficiently hard" environment to compare the synchronization models.

IV. THE GATHERING PROBLEM

In general, the Gathering task requires robots to reach a common placement, not known in advance. This might be a point in the Euclidean plane or a node of a graph. If more

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

than one robot occupies the same location, then we say that a *multiplicity* occurs. The Gathering task is easily solvable once a configuration is reached without pending moves and with exactly one multiplicity, detectable by the robots. In fact, all the robots not composing the multiplicity can move *cautiously* toward it, that is, they move as soon as their trajectory does not encounter other robots.

Solving the Gathering problem depends on the capabilities assumed for the robots. According to the basic capabilities introduced before and to the the setting defined in [21], here robots are considered to be:

- *Autonomous*: no centralized control;
- *Disoriented*: No common coordinate system, no common left-right orientation;
- *Anonymous*: no unique identifiers;
- *Homogeneous*: they all execute the same *deterministic* algorithm;
- *Dimensionless*: no occupancy constraints, no volume;
- *Silent*: no means of direct communication;
- *Stateless (Oblivious)*: no memory of past events;
- *Asynchronous*: ASYNC LCM cycles;

As in [21], in this work we consider the Gathering problem where robots are located on a ring of n nodes, where initially $k < n$ nodes are occupied by k robots. Moreover, during the Look phase, robots can detect multiplicities, but not the exact number of robots composing them.[1]

During a Look operation, a robot basically acquires a *snaphsot* of the environment. More specifically, it perceives the relative locations on the ring of multiplicities and single robots. The global status of the system can be defined by the current disposal of the robots plus their status, that is, whether they are inactive or they are performing the Look, the Compute or the Move phase, and what they have possibly already computed. Clearly, the global status of the system cannot be deduced from the snapshot, including whether there is a pending move or not. It is a common assumption, in the graph environment, that within one move a robot can either stay still or reach a neighboring node, that is only one edge per move can be traversed. Moreover, moves are instantaneous, that is robots are always perceived on nodes and not on edges.

The current disposal of the robots, referred to as a *configuration*, can be described in terms of the *view of a robot* r. This is the sequence of single robots, multiplicities and empty nodes (represented by characters '1', 'M', and '0', respectively) seen by r starting from its position and proceeding toward an arbitrary direction. For instance, by referring to Figure 3, the view of the robot denoted as x can be represented by the sequence $[1,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0]$ (or $[1,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0]$, resp.) obtained starting from x and proceeding in the clockwise (anti-clockwise, resp.) direction.[2] Once a robot has acquired the snapshot of the configuration during the Look phase, in terms of the view, it can process the obtained sequence during the Compute phase in order to infer useful information.

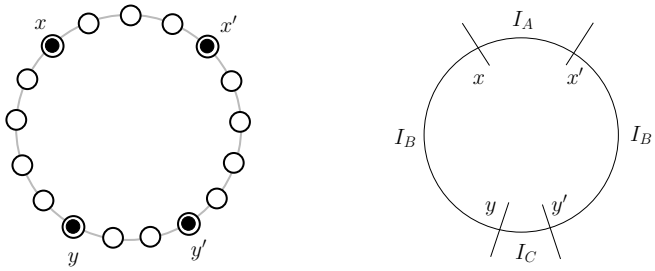


FIGURE 3: A configuration in $\mathcal{SP}4$ and its illustration in terms of intervals of nodes not occupied by robots.

A configuration is said to be *symmetric* if the ring admits a geometrical axis of symmetry that reflects single robots into single robots, multiplicities into multiplicities, and empty nodes into empty nodes; it is said *periodic* if it is invariant with respect to a rotation of more than 0 and less than 360 degrees. We remind that robots are assumed to be disoriented, that is they do not share chirality.

The Gathering problem of robots moving on rings has been almost completely characterized in [21]. In order to describe such a characterization, we first remind that a configuration C is said to be *initial* if there are no multiplicities. Symbol $\mathtt{I}$ is used to denote the set of all initial configurations.

The following negative result has been provided.

Theorem IV.1. *[21] The Gathering problem is unsolvable (even for* SSYNC *robots) with respect to any configuration $C \in \mathtt{I}$ satisfying at least one of the following conditions:*

- *there are only two robots in C;*
- *C is periodic;*
- *C is symmetric admitting an axis of symmetry that cuts the ring along two edges.*

In the remainder, we call *unsolvable* any configuration C as characterized by Theorem IV.1, and use symbol $\mathcal{U}$ to denote the set of all the unsolvable configurations.

In [21], a distributed algorithm has been designed that solves the Gathering problem from all the configurations in $\mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$, where $\mathcal{SP}4 \subset \mathtt{I}$ is a very special subset of initial configurations. To define $\mathcal{SP}4$, consider four robots disposed on different nodes of a ring. As shown in Figure 3, the robots partition the ring into four intervals. An interval is intended as the *maximal set of consecutive nodes not occupied by robots between two robots*; two adjacent robots generate an empty interval.

[1]This is usually referred to as the *global weak multiplicity detection* capability.

[2]Robots have no common orientation which means the 'clockwise' direction of a robot does not necessarily coincide with that of another robot.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

Definition IV.2. *Let $C \in \mathtt{I}$ be a symmetric configuration with four robots on a ring composed by an odd number of nodes (i.e., the axis of symmetry passes through one unoccupied node and one edge). If the interval constituted by an odd number of nodes cut by the axis of symmetry is bigger than the one constituted by an even number of nodes, then C is said to belong to the set of $\mathcal{SP}4$ configurations.*

By looking at Figure 3, there are two intervals cut by an axis of symmetry, namely I_A and I_C, consisting of three and two nodes, respectively. The other two intervals, both referred to as I_B, have the same size since the configuration is symmetric. Further sample configurations in $\mathcal{SP}4$ are shown in Figure 5, configurations (1) and (2), and in Figure 7, configurations (1), (2), (3), and (4).

The only initial configuration of four robots on a five-node ring belongs to $\mathcal{SP}4$. In [28], it has been proved that such a configuration is solvable by means of FSYNC robots, whereas it is unsolvable in both SSYNC and ASYNC. Actually, all the four moves that can be designed (and their compositions) are shown to lead either to two multiplicities or to cyclic sequences of configurations. Clearly, we can deduce the same result of unsolvability also for SASYNC robots as any execution in the SSYNC model can occur in the SASYNC model as well, in fact SSYNC $\geq$ SASYNC.

Corollary IV.3. *The Gathering problem is unsolvable for* SASYNC *robots with respect to the configuration $C \in \mathcal{SP}4$ consisting of four robots on a five-node ring.*

Indeed, specific configurations in $\mathcal{SP}4$ could be solvable but they may require specific strategies not prone to be generalized. One of the main difficulties arising when approaching $\mathcal{SP}4$ configurations is due to the property by which the bigger interval cut by the axis of symmetry is the odd one. In fact, by the result shown in [32], it is known that the only node candidate to finalize the Gathering in configurations in $\mathcal{SP}4$ is the middle node of the odd interval cut by the axis. This remains true also for robots in the FSYNC model. It turns out that robots must move toward such a node, eventually, in order to create a multiplicity. For instance, if robots x and x' or y and y' of Figure 3 are allowed to move, by the algorithm, in the up direction, the adversary may make only one of them moving. Then, if the intervals cut by the axis of symmetry differ of just one node, the obtained configuration is made of two intervals of the same size, those that were originally cut by the axis of symmetry, and hence it is symmetric. However, the new axis of symmetry is different from the original one. This may cause many troubles if also pending moves occur.

Proving that initial configurations with four robots on a seven-node ring are unsolvable in ASYNC has been challenging, since exploring exhaustively all the possible moves becomes computationally intractable. In fact, in [30], both theoretical and computer-assisted analysis have been exploited to prove such a claim. In [28], instead, it has been proved that four robots on a seven-node ring are solvable in SSYNC.

In Table 1, we report the current state-of-art with respect to the Gathering problem for configurations in $\mathcal{SP}4$. From such results we can deduce the following relations: FSYNC $>$ SSYNC $>$ ASYNC. This means that FSYNC robots are more powerful (i.e. they can solve more tasks) than SSYNC robots that in turn are more powerful than ASYNC robots.

FSYNC	SSYNC	ASYNC
$n \geq 5$ [21], [28]	$n = 5$ [28]	$n = 5$ [45]
	$n = 7$ [28]	$n = 7$ [30]
	$n \geq 9$?	$n = 9$ [30]
		$n \geq 11$?

TABLE 1: State-of-art about the Gathering problem for configurations in $\mathcal{SP}4$. Unsolvable cases are reported in gray cells, whereas question marks refer problems that are still open.

Configurations in $\mathcal{SP}4$ for ASYNC robots have been also investigated in [2]. However, in that paper the proposed algorithm only deals with a proper subset of the possible initial configurations, i.e. the authors overcome the main difficulties faced with $\mathcal{SP}4$ configurations by simply ignoring some of them. Hence, this still leaves open the question whether configurations in $\mathcal{SP}4$ are in general solvable for ASYNC robots or not.

A. GATHERING FOR $\mathcal{SP}4$ CONFIGURATIONS

In this section, we provide notation and definitions about the Gathering problem for $\mathcal{SP}4$ configurations.

Given an initial asymmetric configuration, according to its view, a robot can recognize its own position in the ring. In fact, from the snapshot the robot knows the current configuration and where it is placed with respect to the other robots. In an initial symmetric configuration, instead, a robot may have two possible choices for its current position. For instance, referring to Figure 3 and to Figure 5.(1), (2) and (3), robots denoted x and x' are indistinguishable since they cannot distinguish left from right (i.e., they do not share chirality). In initial symmetric configurations on rings constituted of an odd number of nodes, we will denote by x and x' the two robots closest to the node on the axis, whereas the other two robots will be denoted y and y'. Such naming is only meant for the analysis purpose but, as already observed, x is indistinguishable from x' and y is indistinguishable from y'.

The set of consecutive nodes not occupied by robots in between x and x' is referred to as interval I_A. The interval of free nodes between y and y' is denoted by I_C. The interval of free nodes between x and y, and that between x' and y' are both denoted by I_B as they have the same size.

Reminding that during a move a robot can traverse at most one edge, we make use of some additional notation in the designed algorithms: from symmetric configurations, we denote by $r\uparrow$ a move of a robot r toward the middle node

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

of I_A, while by $r\downarrow$ we denote the move in the opposite direction.

In the unique initial asymmetric configuration of four robots on a seven-node ring reported in Figure 5.(4), robots are referred to as a, b, c, and d. We denote by $r \rightarrow r'$ the move of robot r along the shortest path toward robot r' (it is worth noting that in a ring composed by an odd number of nodes, such a direction is unique).

Concerning the already used notion of *symmetric robots*, here we extend it into the more general version of *equivalent robots*. The latter holds in any configuration defined on any graph. In particular, in a configuration C defined on any graph G, a pair of distinct robots r and r' on nodes v and v', respectively, are *equivalent* if there exists a graph isomorphism φ from G into itself that maps v into v', and any pair of equivalent nodes (including v and v') in the isomorphism φ must be occupied by the same number of robots. This equivalence relation induces a partition on the set of all robots: an element of this partition is any maximal subset containing pairwise equivalent robots. For instance, in Figure 4 it is shown a configuration whose robots are partitioned into three sets: one set contains the four equivalent robots located on the pendant nodes with multiplicity two, another set contains the robot adjacent to the empty node, and the last set contains the remaining two robots. Referring to configurations (i), (ii) or (iii) of Figure 5, robots denoted as x and x' are equivalent as well as robots denoted as y and y'. From an algorithmic view

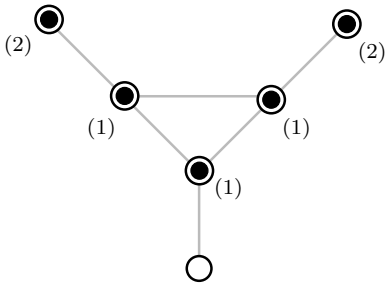


FIGURE 4: A configuration with seven robots defined on a graph (numbers refer to the multiplicity on each node).

point, in such a case it can be observed that the robots belonging to the same maximal subset of equivalent robots cannot be distinguished. As a consequence, no algorithm can avoid that all such equivalent robots perform a same move simultaneously (whereas the adversary may decide not to activate all of them). Finally, we need to define what is meant by a *transition* among configurations. Given an algorithm $\mathcal{A}$ and a configuration C (possibly admitting pending moves), a transition from C to a configuration C' occurs if C' can be obtained during an execution of $\mathcal{A}$ that from C generates C' after at least one robot has moved. Note that the performed movement might be generated by $\mathcal{A}$ from C or due to pending moves.

V. RESULTS FOR SEMI-SYNCHRONOUS ROBOTS

In this section, we provide results for semi-synchronous robots. It is worth reminding that when dealing with synchronous environments like FSYNC or SSYNC, instead of SASYNC or ASYNC, pending moves cannot occur, i.e. robots cannot be seen while they are moving nor even while they have already computed the move but not started it.

We first propose Algorithm 1 referred to as SP4-SSYNC, that takes as input a configuration in $\mathcal{SP}4$, excluding the case of five-node rings, and outputs either a configuration with one multiplicity, or an asymmetric configuration, eventually. Then, such an algorithm is used as a subroutine for the Gathering algorithm proposed in [21]. The resulting algorithm obtained as the composition of SP4-SSYNC with the algorithm in [21], not only solves the problem for $\mathcal{SP}4$ configurations, but also provides a full characterization of the Gathering problem in the SSYNC model.

Algorithm 1 SP4-SSYNC. A move of a robot r toward the middle node of interval I_A (cf. Figure 3, right side) is denoted by $r\uparrow$.

Require: Configuration in $\mathcal{SP}4$ on a ring of $n \geq 7$ nodes.
Ensure: Gathering.
1: **if** $(|I_A| > |I_C|+1) \vee (|I_B|$ is odd$) \vee (x$ and y are adjacent)
2: **then** $x\uparrow$
3: **else** $y\uparrow$

Before proving the correctness of SP4-SSYNC, we describe an example of execution just as a warm-up for the reader.

Consider configuration (2) of Figure 5 as input. It belongs to $\mathcal{SP}4$. Recalling from Figure 3 how configurations in $\mathcal{SP}4$ can be interpreted, here we have $|I_A| = |I_C|+1$. Moreover, x and y are not adjacent. However, $|I_B| = 1$, i.e. it is odd. The move dictated by algorithm SP4-SSYNC is then $x\uparrow$. Being the configuration symmetric, the move allows both x and x' to move. If they both move concurrently, a multiplicity is created and the algorithm terminates. If without loss of generality, only x moves, then configuration (3) of Figure 5 is obtained. From there we have $|I_A| = |I_C| + 1$, $|I_B| = 0$ but x and y are adjacent. Hence, algorithm SP4-SSYNC says $x\uparrow$. Again, if both x and x' move concurrently, then a multiplicity is created. Otherwise, the asymmetric configuration (4) of Figure 5 is obtained.

Lemma V.1. *Given a configuration $C \in \mathcal{SP}4$ on rings of $n \geq 7$ nodes, Algorithm* SP4-SSYNC *moves robots so that, within a finite number of moves, the obtained configuration C' either contains one multiplicity or $C' \in \mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$.*

Proof. Assume $|I_A| > |I_C| + 1$ and consider move $x\uparrow$ at Line 2. If, without loss of generality, only x moves, then the obtained configuration C' is asymmetric. In fact, being the ring composed by an odd number of nodes, there cannot be an axis with robots. So, any possible axis should cut two intervals among those of resulting size $|I_A|-1$, $|I_B|+1$, $|I_C|$ and $|I_B|$. Clearly, the axis cannot cut intervals with $|I_A|-1$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

and $|I_C|$ nodes, as the other two intervals have different size. There cannot be an axis passing through intervals with $|I_B|+1$ and $|I_B|$ nodes because $|I_A|-1 > |I_C|$. Being asymmetric, C' is not in $\mathcal{SP}4$.

If both x and x' move, still a configuration in $\mathcal{SP}4$ is obtained but with interval I_A decreased of two units, and intervals I_B increased by one. In this way, within a finite number of moves, either an asymmetric configuration or a configuration with $|I_A| = |I_C|+1$ is obtained.

We can now assume $|I_A| = |I_C|+1$. If $|I_B|$ is odd, then we have to analyze what happens performing $x\uparrow$. If both x and x' move, the obtained configuration maintains the original axis of symmetry but it is not in $\mathcal{SP}4$ as $|I_A|-2 < |I_C|$. Moreover, a multiplicity is created in case $|I_C| = 0$. If only x moves, $|I_A|-1 = |I_C|$ and there is an axis passing through intervals of size $|I_B|$ and $|I_B|+1$. Even though it is a symmetric configuration, it is not in $\mathcal{SP}4$ as, by assumption, $|I_B|$ is odd and of course it is smaller than $|I_B|+1$.

We can now assume $|I_B|$ even. If x and y are adjacent, then we have to analyze what happens performing $x\uparrow$. As above, if both x and x' move, the obtained configuration is not in $\mathcal{SP}4$ and a multiplicity is created in case $|I_C| = 0$. If only x moves, $|I_A|-1 = |I_C|$ and there is an axis of symmetry passing through intervals of size $|I_B|$ and $|I_B|+1$. Now, the obtained configuration $\tilde{C}$ is in $\mathcal{SP}4$ because $|I_B|$ is even. With respect to $\tilde{C}$, we are in the case where $|I_A| = |I_C|+1$, $|I_B|$ is even, but x and y are not adjacent as the ring has at least seven nodes.[3]

We can now assume that x and y are not adjacent. In this case, move $y\uparrow$ is applied at Line 3. If both y and y' move, a configuration with intervals with $|I_A|$, $|I_B|-1$, $|I_B|-1$ and $|I_C|+2$ nodes is achieved, which is not in $\mathcal{SP}4$ since now $|I_A| = |I_C|-1$ (it was $|I_A| = |I_C|+1$ before the moves). If, without loss of generality, only y moves, then the obtained configuration admits an axis of symmetry passing through the two intervals of size $|I_B|$ and $|I_B|-1$ since the interval of size $|I_C|$ has been increased of one unit and $|I_C|+1 = |I_A|$. Since $|I_B|$ is even, the obtained configuration is not in $\mathcal{SP}4$.

Finally, it is easy to check that each configuration obtained in the analyzed cases is not in $\mathcal{U}$ since, for each of them, none of the cases of Theorem IV.1 applies. □

Reminding the result of [21] concerning ASYNC robots, we can then exploit the above lemma in order to obtain a full characterization of the Gathering problem on rings in the SSYNC model. In fact:

Theorem V.2. *[21] The Gathering problem on rings can be solved with* ASYNC *robots with respect to any configuration in* $\mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$ *without ever leading to a configuration in* $\mathcal{SP}4$.

[3]We remind that $n > 5$ is required by the fact that the unique initial configuration of four robots on a five-node ring is unsolvable even in SSYNC.

Since Lemma V.1 ensures that any configuration in $\mathcal{SP}4$ of SSYNC robots, on a ring of $n \geq 7$ nodes, can be transformed either into a configuration with one multiplicity or into a configuration in $\mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$, within a finite number of moves, then, in order to obtain a general Gathering algorithm for SSYNC robots on rings, we can distinguish two cases: i) the input configuration is in $\mathcal{SP}4$ on a ring of $n \geq 7$ nodes; ii) the input configuration is in $\mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$.

In case i), the resolution algorithm can apply Algorithm SP4-SSYNC to obtain either a configuration with a multiplicity or a configuration in $\mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$. From there, the configuration can be managed by the algorithm in [21]. In fact, such an algorithm is able to manage the configurations with exactly one multiplicity and by Theorem V.2, it copes with all the configurations in $\mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$ of ASYNC (and hence also of SSYNC) robots without ever leading to a configuration in $\mathcal{SP}4$.

In case ii), the algorithm in [21] can be applied straightforwardly.

Summarizing, by combining Algorithm SP4-SSYNC with that in [21], we obtain that the Gathering on rings can be accomplished by SSYNC robots starting from any configuration which has not been proved to be unsolvable. More specifically:

Theorem V.3. *The Gathering problem on rings can be solved by* SSYNC *robots with respect to any configuration* C *if and only if* $C \in \mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$ *or* $C \in \mathcal{SP}4$ *on a ring of* $n \geq 7$ *nodes.*

Proof. ($\Longrightarrow$) If C belongs to $\mathcal{U}$, it is clearly unsolvable by Theorem IV.1. If C belongs to $\mathtt{I} \setminus \mathcal{U}$ and it is in $\mathcal{SP}4$ with $n < 7$ nodes, then we get exactly the only initial configuration of four robots on a five-node ring. According to [28], such a configuration is unsolvable in SSYNC.

($\Longleftarrow$) Let us first assume that C belongs to $\mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$. To show that C is solvable, it is enough to consider the algorithm provided in [21] designed for Gathering any configuration in $\mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$ by means of ASYNC robots. Clearly, it can be applied also in the SSYNC context. Hence, by Theorem V.2, C is solvable without ever leading to a configuration in $\mathcal{SP}4$.

If $C \in \mathcal{SP}4$ on a ring of $n \geq 7$ nodes, then we can combine the algorithm provided in [21] with Algorithm SP4-SSYNC. In fact, according to Lemma V.1, Algorithm SP4-SSYNC handles C in order to obtain either a configuration C' with one multiplicity (and without pending moves) or a configuration $C' \in \mathtt{I} \setminus (\mathcal{U} \cup \mathcal{SP}4)$. From C', the algorithm in [21] used in the SSYNC context guarantees to finalize the Gathering. □

Notice that the above result provides a full characterization of the Gathering problem on rings in the SSYNC model.

VI. RESULTS FOR SEMI-ASYNCHRONOUS ROBOTS

This section is mainly devoted to prove the inequalities SSYNC > SASYNC > ASYNC. We first show that inequality

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

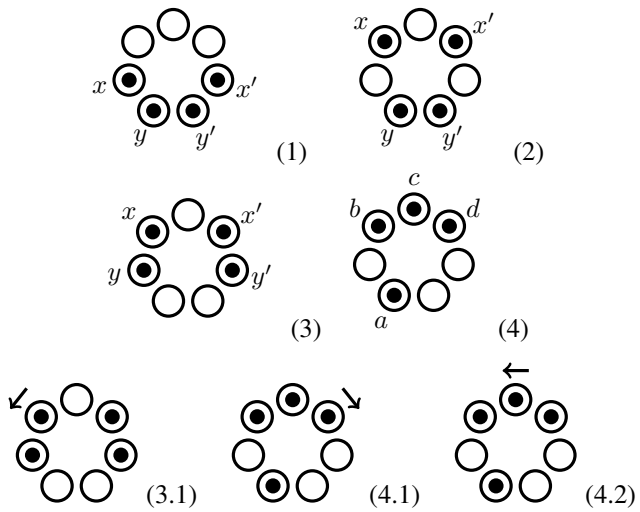


FIGURE 5: The four possible initial configurations of four robots on a seven-node ring, and other three configurations with possible pending moves. Edges composing the ring between consecutive nodes are not drawn. An arrow on top of a robot represents a pending move, that is a move already computed by the robot that has not yet been performed.

SASYNC $>$ ASYNC holds by proving that configurations of four robots on rings of $n = 7$ nodes can be always solved in SASYNC, whereas it is known that those belonging to $\mathcal{SP}4$ cannot be solved in ASYNC (see [30]). Then, we show that inequality SSYNC $>$ SASYNC holds by proving that configurations in $\mathcal{SP}4$ with $n = 9$ cannot be solved in SASYNC, whereas in Section V we have shown that such configurations are solvable in SSYNC. Finally, we also present an algorithm for SASYNC robots that solves all the configurations of four robots on rings of $n = 11$ nodes. Such a case remains open in the ASYNC model.

For the case $n = 7$, Algorithm 2 referred to as $4on7$-SASYNC brings SASYNC robots to a configuration with one multiplicity within a finite number of moves. From there, the Gathering can be then easily solved.

Algorithm 2 $4on7$-SASYNC.

Require: Configuration C on a seven-node ring.
Ensure: Gathering.
1: **if** C contains a multiplicity m **then**
2: the robot(s) closest to m, but not on it,
3: moves toward m along the shortest path
4: **else**
5: **if** C is symmetric **then** $x\uparrow$
6: **else** $d \rightarrow c$;

Theorem VI.1. *The Gathering problem can be solved for* SASYNC *robots with respect to any initial configuration C with four robots on seven-node rings.*

Proof. Figure 5, cases (1)–(4), shows all the possible configurations in I with four robots on seven-node rings that

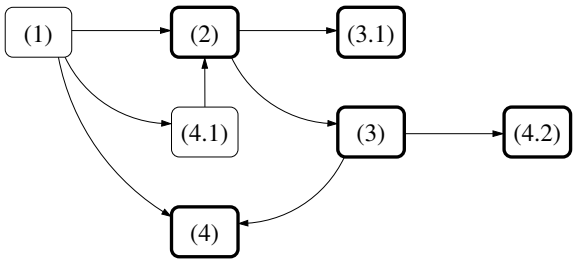


FIGURE 6: Possible transitions among configurations without multiplicities produced by Algorithm $4on7$-SASYNC. Nodes in the graph represents any possible configuration handled by Algorithm $4on7$-SASYNC (for node labels refer to Figure 5), while nodes with thick border represent configurations leading to a multiplicity.

Algorithm $4on7$-SASYNC can take as input, with (1) and (2) belonging to $\mathcal{SP}4$. We now analyze the behavior of the algorithm with respect to all the possible handled configurations (cf. Figure 5).

- From (1), the algorithm makes $x\uparrow$. This may lead to three different cases: (2), (4), or (4.1) where the move of a robot is pending.
- From (2), $x\uparrow$ is applied. If both the equivalent robots move, a multiplicity is created without pending moves, hence Gathering can be easily finalized. If only one robot moves, the other may be pending or not, hence reaching configurations (3) or (3.1), respectively.
- From (3) the algorithm makes $x\uparrow$. If both robots move, a multiplicity is created. If only one robot moves, configurations (4) or (4.2) are possibly reached.
- From (4) the algorithm makes robot d move toward robot c, hence creating a multiplicity. From there, the Gathering can be finalized.
- From (3.1), the algorithm makes $x\uparrow$. Within at most two phases of the LCM cycle, the pending move will create a multiplicity, whereas the other robot, possibly moving with respect to $x\uparrow$, does not interfere with such an event since its move is 'compatible' with the movement toward the multiplicity, that is such a move is toward the multiplicity and does not create other multiplicities. Notice that, by the SASYNC model, the pending move is performed before a subsequent move takes place.
- From (4.1), d would move in the opposite direction with respect to that specified by the algorithm, but no other robot moves. This leads to (2).
- From (4.2), as noted above, the pending robot will always create a multiplicity before the move $d \rightarrow c$ applied in (4.2) is completed. Also in this case the move $d \rightarrow c$ dictated by the algorithm from (4.2) is compatible with the movement toward the multiplicity.

Summarizing, Figure 6 shows all the transitions (represented as directed arcs) among the configurations handled by Algorithm $4on7$-SASYNC.

From Figure 6, it is easy to see that Algorithm $4on7$-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

SASYNC never creates cycles among configurations and that it eventually creates a multiplicity starting from any input configuration. Notice that, possibly pending moves are left but they are compatible with the movement toward the multiplicity where the Gathering will be finalized, that is, they do not affect the correctness of the algorithm. □

Theorem VI.2. *The Gathering problem is unsolvable for* SASYNC *robots with respect to any initial configuration* $C \in \mathcal{SP}4$ *with nine-node rings.*

Proof. We now prove the claim by defining a specific behavior of the adversary when the configuration does not contain multiplicities: *Starting from one configuration in* $\mathcal{SP}4$*, whatever a Gathering algorithm specifies to move, the adversary always allows the robots belonging to one maximal subset of equivalent robots to move. In particular, it allows synchronous moves as long as the configuration remains in* $\mathcal{SP}4$ *or become unsolvable, otherwise only one robot is allowed to move, possibly leaving a pending move.*

It is easy to see that in nine-node rings, there are 10 possible initial configurations, 6 of which are symmetric. Out of the 6 symmetric configurations, 4 are $\mathcal{SP}4$ configurations (see Figure 7, configurations (1)–(4)).

Before proceeding with the proof, we need to observe that a configuration composed of just two multiplicities can be thought as equivalent to a configuration with just two robots, i.e. it is unsolvable according to Theorem IV.1. In fact, the adversary can make all the robots composing a multiplicity move synchronously.

We now analyze the behavior of any possible Gathering algorithm with respect to all the possible handled configurations (cf. Figure 8), considering that only one maximal subset of equivalent robots can move according to the considered adversary. Note that, this does not mean that the hypothetic Gathering algorithm must specify only one maximal subset of equivalent robots to move from a given configuration. However, among the maximal subsets involved, the adversary will allow only one to move, not necessarily the same subset from the same configuration as this might be subject to wait-freedom constraints. By proving that for any maximal subset of moving robots the reached configurations are either unsolvable or belong to $\mathcal{SP}4$, provides the proof as this does not depend on the behavior of the adversary and its wait-freedom constraint.

- From (1), only $x\uparrow$ can make the configuration evolve as any other move would lead to either an unsolvable configuration with two multiplicities ($x\downarrow$ and $y\uparrow$ cases), or to an infinite loop ($y\downarrow$ case) remaining in (1). By the defined adversary, configuration (2) is then reached.
- From (2) only two moves can be allowed by any Gathering algorithm, that is $x\uparrow$ and $y\uparrow$ that lead to (3) and (4), respectively. In fact, $x\downarrow$ would lead back to (1), creating a loop, whereas $y\downarrow$ would leave the configuration unchanged in (2).
- From (3), the only ways to exit $\mathcal{SP}4$ are either by $x\uparrow$ or $y\uparrow$. Concerning $x\uparrow$, by making move only one robot between x and x', configuration in (4) is again obtained. Concerning $y\uparrow$, configuration in (5) is obtained either by moving only one robot or by moving both y and y' but with the pending move of y', as shown in (5.1). The actual move is chosen by the adversary according to the move scheduled by the Gathering algorithm with respect to case (5).
  Notice that, since we are considering SASYNC robots, the pending move in (5.1) cannot last for long as in the ASYNC model, but we are guaranteed it will be performed within a time unit after it has been computed.
  The other possible moves are $x\downarrow$, which would lead back to (2) thus creating a loop, or $y\downarrow$, that would leave the configuration unchanged in (3).
- From (4), the only way to exit $\mathcal{SP}4$ is by $x\uparrow$, in which case the adversary makes only one robot move, hence leading to (3). Concerning the other possible moves: both $y\uparrow$ and $x\downarrow$ would lead to an unsolvable configuration with two multiplicities, whereas $y\downarrow$ would lead back to (2).
- From (5), let us analyze any possible move performed by the Gathering algorithm. Recall that (5) is not in $\mathcal{SP}4$ but it could be generated from (3), either as (5) or with the pending move in (5.1). Then:
  - -- if the algorithm applies $x\uparrow$, the adversary brings the configuration to (3) starting from (5.1) and performing both the move of the pending robot and $x\uparrow$ (where $x\uparrow$ moves only one robot). Note that the adversary can do that by awaking the robot that would apply $x\uparrow$ before the pending move is performed. It follows that move $x\uparrow$ from (5) is not effective;
  - -- if the algorithm applies $x\downarrow$, the adversary brings the configuration to (4) by starting from (5);
  - -- if the algorithm applies $y\downarrow$, the adversary brings the configuration to (3) by starting from (5);
  - -- if the algorithm applies $y\uparrow$, the adversary brings the configuration to (5.2) by starting from (5.1), performing the pending move and leaving pending $y\uparrow$ which now corresponds to $x\uparrow$. Then, from (5.2), by performing both the pending move $x\uparrow$ and the move $y\uparrow$, again (2) is obtained.

In conclusion, we have exhaustively considered all possible moves (and implicitly also their combinations) from all possible initial configurations belonging to $\mathcal{SP}4$, hence showing there exists no strategy that ensures to reach a configuration outside those in $\mathcal{SP}4$. □

We now provide Algorithm 3, referred to as $4on11$-SASYNC, to solve the Gathering problem from configurations on eleven-node rings with four robots. The aim is to show the impact of the SASYNC model on the resolution

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

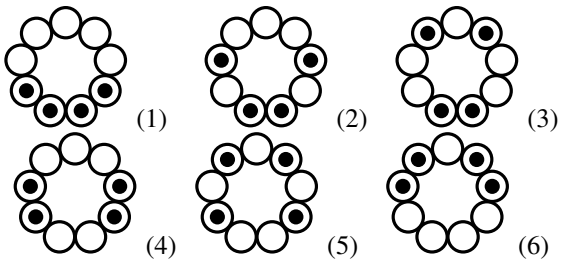


FIGURE 7: The possible initial configurations of four robots on nine-node rings that are symmetric. Configurations (1)–(4) belong to $\mathcal{SP}4$.

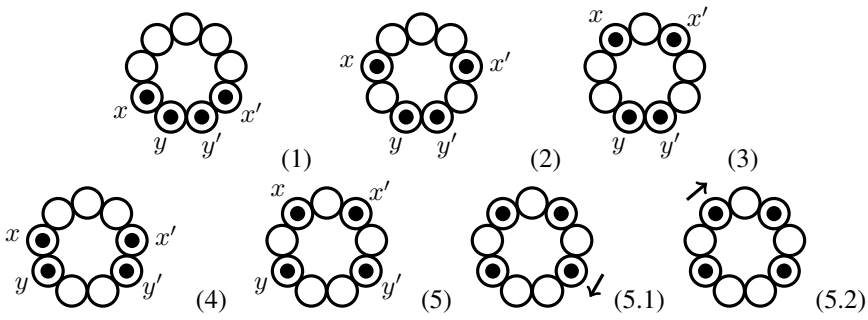


FIGURE 8: The four possible initial configurations belonging to $\mathcal{SP}4$ of four robots over a nine-node ring and a configuration not in $\mathcal{SP}4$ with two possible pending moves (cf. proof of Theorem VI.2).

of the problem when $n = 11$ that actually remains open in ASYNC.

Theorem VI.3. *The Gathering problem can be solved for* SASYNC *robots with respect to any initial configuration C with four robots on eleven-node rings.*

Proof. Figures 9 and 10 show all the possible configurations in $\mathtt{I}$ with four robots on an eleven-node ring. In particular, Figure 9 shows all the initial symmetric configurations (out of which the first six belong to $\mathcal{SP}4$), while Figure 10 shows all the initial asymmetric configurations. Concerning the asymmetric configurations, according to Figure 10, the robots encountered on the depicted rings, starting from the leftmost one and proceeding in the anti-clockwise direction, are denoted by a, b, c, d.

Algorithm $4on11$-SASYNC is designed to solve the Gathering problem from all those configurations.

The proof about the correctness of Algorithm $4on11$-SASYNC is done by an exhaustive case-by-case analysis of the configurations that can be generated and that in fact lead to Gathering. In order to understand it, we need to specify some notation.

Configurations denoted by a decimal number – e.g., configuration (3.1), see Figure 11 – concern those with pending moves that may occur according to the defined algorithm. In Figure 12, any node with thick border represents a case in which there could be a transition leading to the creation of a multiplicity. For the ease of notation, such transitions are simply omitted. Moreover, in such cases there might still be pending moves by means of robots not belonging to the multiplicity, however it can be checked they do not interfere with the finalization of the Gathering.

According to the algorithm, it can be checked that all the possible transitions among configurations are those shown (or implicitly represented) in Figure 12. Configurations (12), (13) and (15) do not appear in the figure as, from each of them, the algorithm would generate a multiplicity just after the first move, whereas no transition leads to them.

From the discussion above and from the analysis of the transitions shown in Figure 12, it can be observed that algorithm $4on11$-SASYNC is able to solve the Gathering problem from each initial configuration. □

We conclude this section by summarizing all the results obtained with respect to the considered synchronization models. This is done by presenting Table 2, which also updates the current status of the Gathering problem in $\mathcal{SP}4$ configurations and, in turn, represents a proof of Theorem III.5. In particular, Table 2 shows a first relevant difference between SASYNC and ASYNC (i.e., SASYNC $>$ ASYNC); it refers to the Gathering problem in $\mathcal{SP}4$ configurations for the case $n = 7$. In fact, in [30] it has been proved that Gathering four robots on seven-node rings is unsolvable for ASYNC robots. The provided proof is rather involving as it exploits both theoretical aspects and computer-assisted evaluations in order to exhaustively explore all possibilities. In SASYNC, instead, Algorithm $4on7$-SASYNC represents a rather easy solution, also with respect to the associated

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

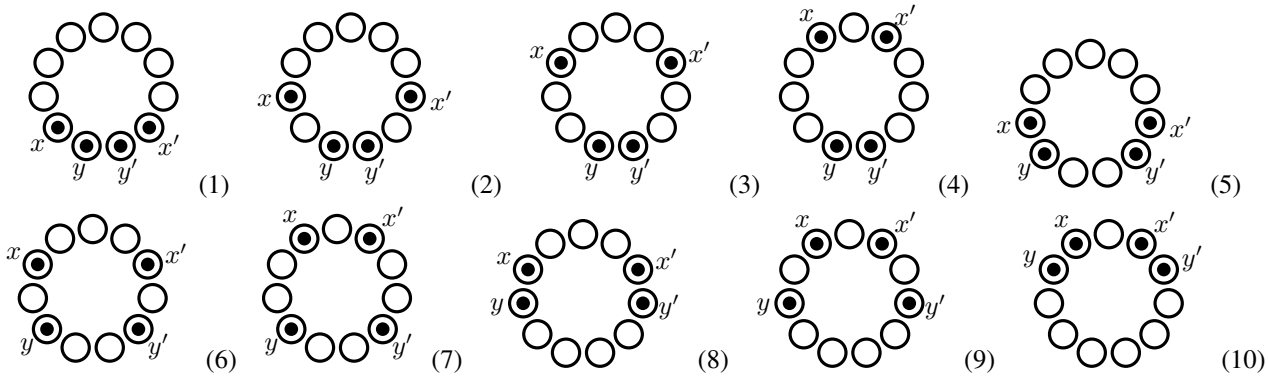


FIGURE 9: The symmetric initial configurations of four robots on eleven-node rings. Note that configurations (1)–(6) belong to $\mathcal{SP}4$.

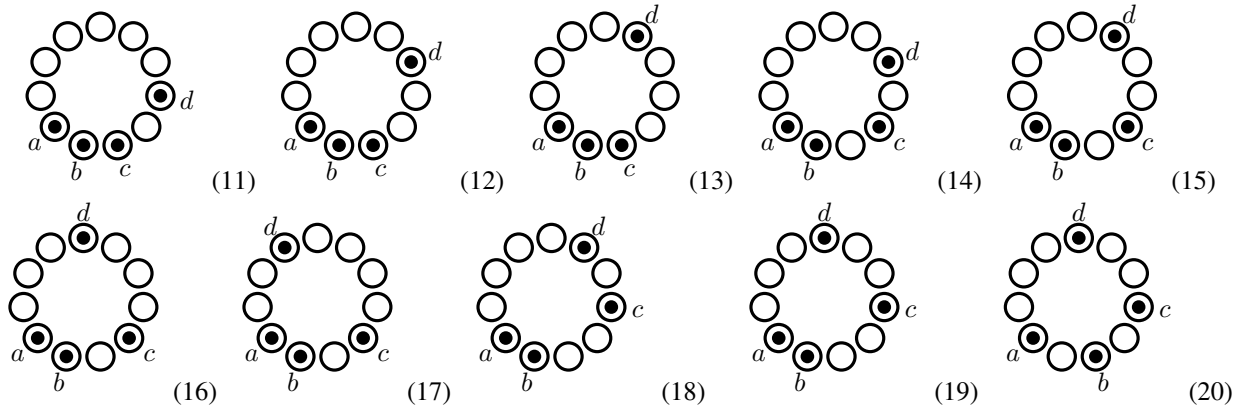


FIGURE 10: The asymmetric initial configurations of four robots on eleven-node rings.

Algorithm 3 $4on11$-SASYNC.

Require: Configuration C on an eleven-node ring.
Ensure: Gathering.

```
if C contains a multiplicity m then
    the robot(s) closest to m, but not on it,
    moves toward m along the shortest path
else
    if C is symmetric then
        if C ≡ (9) then y↑
        else x↑
    else
        if C ≡ (14) then d → c
        else
            if C ∈ {(17), (19)} then c → d
            else
                if C ≡ (20) then c → b
                else a → b
```

proof of correctness.

The other difference, necessary to complete the proof of Theorem III.5, comes out by comparing Theorems V.3 and VI.2. Table 2 shows that such theorems imply SSYNC > SASYNC; this result refers to the Gathering problem in $\mathcal{SP}4$ configurations for the case $n = 9$, solvable in SSYNC and unsolvable in SASYNC.

FSYNC	SSYNC	SASYNC	ASYNC
$n \geq 5$ [21], [28]	$n = 5$ [28]	$n = 5$ **[Cor. IV.3]**	$n = 5$ [45]
	$n \geq 7$ **[Th. V.3]**	$n = 7$ **[Th. VI.1]**	$n = 7$ [30]
		$n = 9$ **[Th. VI.2]**	$n = 9$ [30]
		$n = 11$ **[Th. VI.3]**	$n \geq 11$?
		$n \geq 13$?	

TABLE 2: State-of-art about the Gathering problem for configurations in $\mathcal{SP}4$ after including our results (cf. Tables 1). Unsolvable cases are reported in gray cells, question marks refer to problems that are still open, whereas results obtained within this paper are highlighted in bold.

Finally, the case $n = 11$ solvable in SASYNC according to Theorem VI.3 remains open for ASYNC. Still the result is interesting as it shows there is not a net separation between solvable and unsolvable cases in SASYNC due to the size of the ring.

VII. ABOUT THE EUCLIDEAN PLANE

Similarly to what has been done in [50] to show that SSYNC > ASYNC, here we can prove that SASYNC > ASYNC. In fact, this can be achieved by considering the task referred to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

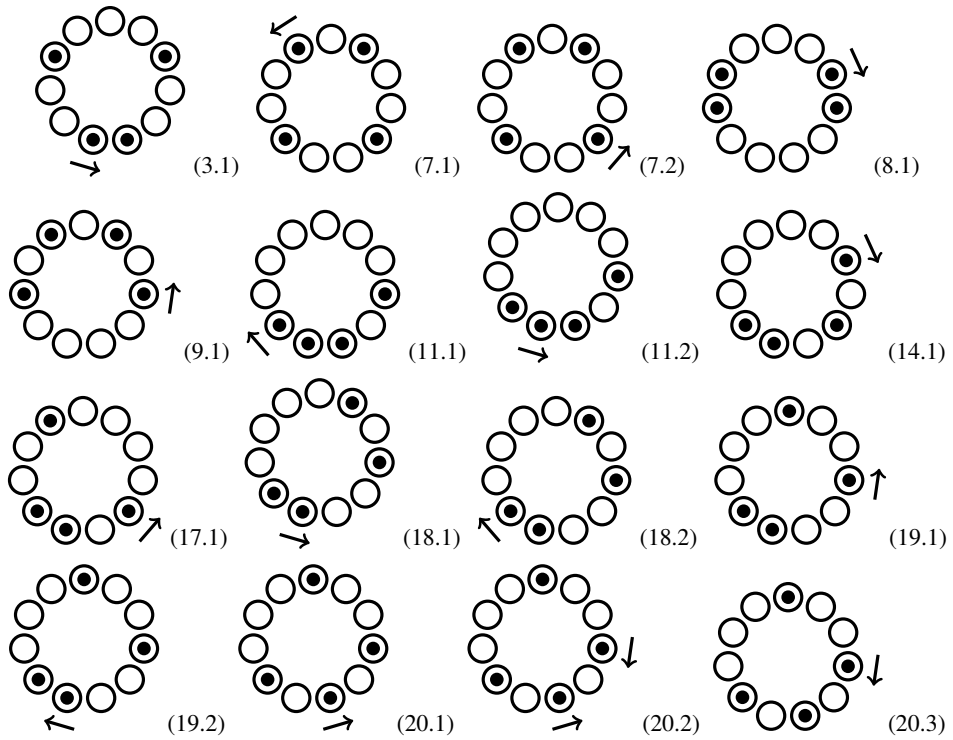


FIGURE 11: All possible configurations with pending moves that can be generated before creating a multiplicity by algorithm $4on11\text{-}SAsync(1)$.

as *Movement Awareness* defined in [50] for robots admitting states (i.e., non-oblivious). Still remains open to establish whether the inequality SSYNC $\geq$ SASYNC is strict also for stateless robots.

In the remainder, we investigate on the computational relation between SASYNC and SSYNC for stateless robots. In particular, we consider the Gathering task for robots moving in the Euclidean plane with exactly the same setting of [16] and [52], that is, the same assumptions as in Section IV plus the following one:

- Non-rigid: robots are not guaranteed to reach a destination within one move;

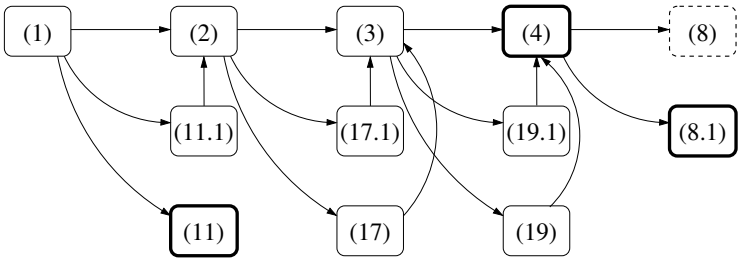
When considering the algorithms provided in the literature to solve the Gathering problem in ASYNC [16] and in SSYNC [52], it leaps out the main difference in the complexity required. In fact, both the design of the algorithms as well as the related proofs of correctness look very different in terms of readability and argumentations. Still, both algorithms solve the Gathering problem starting from any initial configuration but those composed of just two robots. Hence, investigating on Gathering cannot lead to a prove that the inequality ASYNC $\leq$ SASYNC $\leq$ SSYNC is strict as we did in the context of robots moving on graphs. However, we felt to examine in more depth the context of robots moving in the Euclidean plane in order to better understand how SASYNC can be approached. In particular, can we provide a simple algorithm as in the SSYNC case or the difficulties arising in the ASYNC case are already present in SASYNC? From our investigation, it seems that SASYNC does not prevent to introduce the arguments required for ASYNC, hence showing its 'hostile' nature. It follows that, the drastic reduction concerning the obsolescence of the pending moves occurring in SASYNC with respect to ASYNC seems to be not effective when looking for a resolution algorithm in the context of robots moving in the Euclidean plane. As a benefit of our investigation, instead, our outcome seems to suggest that, when dealing with robots moving in the Euclidean plane, it is enough to focus on the SASYNC model rather than approaching the ASYNC one along with all its complications.

Before providing the algorithm designed in [52], we first need some notation. Let R be a multiset of points in the plane. By $C(R)$ and $c(R)$ we denote the smallest enclosing circle of R and its center, respectively. Let C be any circle concentric to $C(R)$. We say that a robot $r \in R$ is *on* C if and only if r is on the circumference of C; ∂C denotes all the robots on C. We say that a robot $r \in R$ is *inside* C if and only if r is in the area enclosed by C but not in ∂C; $int(C)$ denotes all the robots inside C. The smallest enclosing circle $C(R)$ is unique and can be computed in linear time [49].

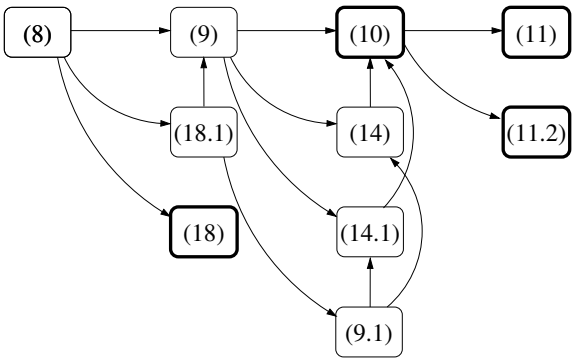
For a fair comparison among the two algorithms, we restrict to the case of configurations composed by $n \geq 5$ robots, initially occupying different positions. In such a setting, Algorithm 4 summarizes the strategy proposed in [52].

Since, by definition, in SSYNC there cannot occur pend-

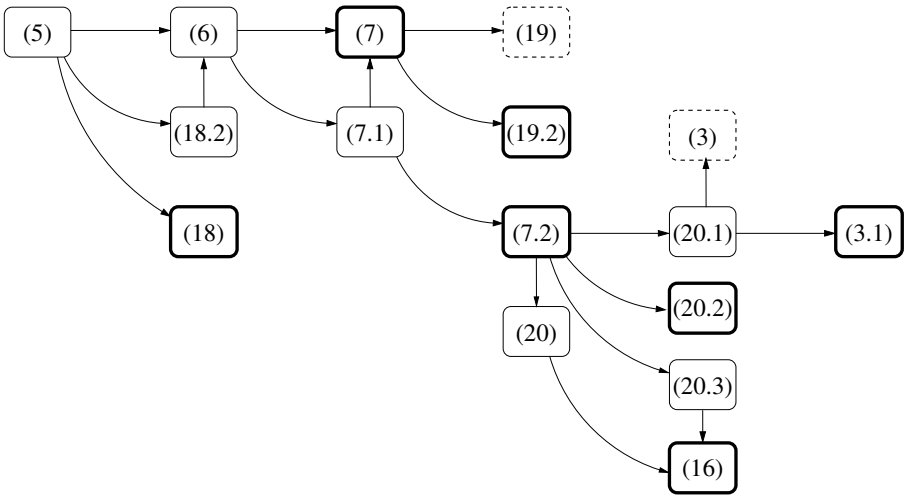
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access



(a) Transitions starting from configurations (8)–(11), (14) and (18).



(b) Transitions starting from configurations (1)–(4), (11), (17) and (19).



(c) Transitions starting from configurations (5)–(7), (16), (18)–(20).

FIGURE 12: Transitions produced by Algorithm $4on11$-SASYNC before creating a multiplicity (for node labels refer to Figures 9, 10, and 11; nodes with thick border represent configurations leading to a multiplicity). Nodes with scattered border represent configurations which already appeared in a preceding sub-figure. For the sake of simplicity, configurations (12), (13) and (15) do not appear in the figure as from each of them the algorithm generates a multiplicity (without pending moves) just after the first move.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

ing moves, then it is quite easy to infer the correctness of Algorithm 4. In fact, before a multiplicity is created, only the three cases caught by Lines 4–9 can occur. Moreover, during the movements, no undesired multiplicities can be created. Once a multiplicity is done, each other robot is cautiously moved toward it, that is without overpassing other robots.

By means of the next lemma, we show that the cases caught by Lines 4–5 represent a strategy necessary also when assuming FSYNC robots.

Algorithm 4 Algorithm for $n \geq 5$ robots in SSYNC [52].

Require: Configuration R with $n \geq 5$ robots.
Ensure: Gathering.
1: **if** there is a multiplicity at point m **then**
2: all robots in R cautiously move toward m
3: **else**
4: **if** $|int(C(R))| = 0$ **then**
5: all robots in R cautiously move toward $c(R)$
6: **if** $|int(C(R))| = 1$ **then**
7: $r \in int(C(R))$ moves toward the closest robot
8: **if** $|int(C(R))| > 1$ **then**
9: robots in $int(C(R))$ cautiously move toward $c(R)$

Lemma VII.1. *Let R be an initial configuration with $n \geq 5$ robots forming a regular n-gon on $C(R)$. Any Gathering algorithm must move robots toward $c(R)$.*

Proof. As the robots form a regular n-gon, they are all equivalent. It means that, any move dictated by a resolution strategy involves all such robots. It follows that the adversary may force all the robots to work synchronously, and hence any move toward a point different from $c(R)$ can be performed by all the robots. This would leave the configuration similar to the initial one. In fact, the robots would keep on forming always a regular n-gon even though the smallest enclosing circle may change. Hence, the claim holds. □

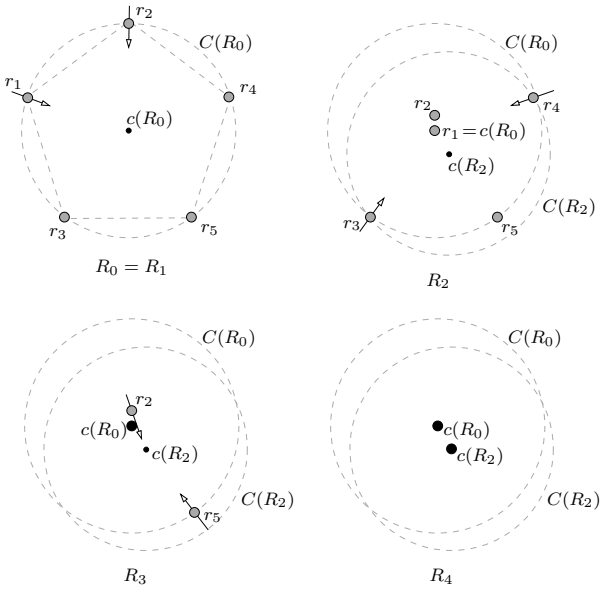
We now consider Algorithm 4 in the context of SASYNC, hence showing the main difficulties arising and why that algorithm does not work. Clearly, Lemma VII.1 also holds in SASYNC. Hence, if the initial configuration is given by n robots forming a regular n-gon on $C(R)$, the resolution strategy must move them toward $c(R)$. However, now pending moves can occur, even though they can stand as such just for one time unit. Unfortunately, this is already enough to face much more difficulties in designing a resolution strategy. We can describe a possible execution $R = R_0, R_1, R_2, \ldots$ (with R_i being the configuration observed at time i) of Algorithm 4 with respect to the SASYNC model where complications arise.

The scenario is described by Figure 13.(a), by means of a configuration $R = R_0$ composed of five robots disposed as a pentagon. As non-rigid movements are assumed, the adversary can make move any subset of robots toward $c(R_0)$ at different distances. This implies that, at the initial time 0, only a subset of robots could be active; moreover, the adversary may prevent such active robots to form a multiplicity in $c(R_0)$ and to obtain a configuration at time 2 where $c(R_2) \neq c(R_0)$. In the case described by Figure 13, the active robots at time 0 are r_1 and r_2, and in R_2 we have that r_1 has reached $c(R_0)$, whereas r_2 has been stopped before. Figure 13.(b) shows the activation schema of all the five robots. With respect to such a schema, it turns out that robots r_3 and r_4 have already computed their moves (that are pending) at time 2. In fact, their target is $c(R_0)$ and it has been computed at time 1 according to the algorithm. However, the computed moves are performed in the subsequent time step, while robots r_2 and r_5 are finalizing their Compute phase on the basis of the snapshot acquired at time 1, that is, the snapshot acquired by r_3 and r_4 reveals configuration $R_1 = R_0$. Hence, at time 3, configuration R_3 may occur in which r_3 and r_4 have formed a multiplicity on $c(R_0)$ where also r_1 resides. Since both r_1 and r_5 perform their Look phase at time 2, they both execute Lines 8–9 of Algorithm 4 (because they both belong to $int(C(R_2))$). It follows that r_2 and r_5 move toward $c(R_2)$ and the adversary can make both complete their movements at time 4. The obtained configuration R_4 then contains two multiplicities, one at $c(R_0)$ and one at $c(R_2)$. From R_4, the adversary may prevent the robots to ever finalize the Gathering since a configuration composed by just two multiplicities is basically equivalent to have just two robots. In fact, the adversary can make move all the robots composing one multiplicity in a synchronous way, i.e., like if they were a unique entity. Since it is known that the Gathering is unsolvable for $n = 2$ even in SSYNC, see [52], then the obtained configuration R_4 is unsolvable. It follows that Algorithm 4 cannot be applied as it is to solve the Gathering problem in SASYNC. In fact, the scenario reported in Figures 13 can be easily extended to configurations with an arbitrary number of robots.

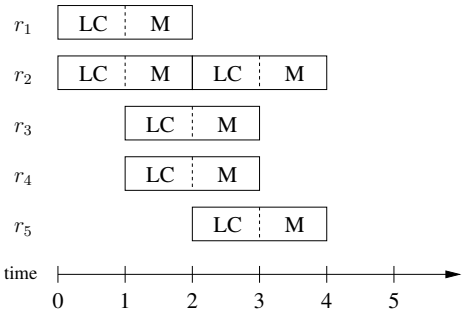
Modifications to the strategy proposed by Algorithm 4 to avoid the difficulty described above are then required. Hypothetically, we may want to find a move that can 'waste time' before moving the robots toward the center of the current smallest enclosing circle. This may allow possible pending moves to be realized, hence either creating only one multiplicity in the original $c(R)$ or letting the robot move successively to the new center. For instance, from R_2 one may think about rotating the robots on $C(R)$ along the circumference toward specific targets that permit to infer the termination of the rotation. If such a movement can be realized, all the robots can deduce from there that no pending moves are standing. Clearly, it is possible that pending moves originated in R_0 are performed and they either create a multiplicity or simply change the number of robots involved by the move designed for R_2. Unfortunately, it seems it is not that obvious to realize such a strategy.

On the other extreme, there is of corse the strategy proposed in [16] dealing with ASYNC robots. This is based on the preliminary step where robots first check whether the current configuration is *equiangular* or not. A configuration

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access



(a) The first four configurations of the execution. Gray circles denote robots, black circles denotes multiplicities, arrows denote movement directions.



(b) The SASYNC activation schema of robots.

FIGURE 13: A visualization of the execution of Algorithm 4 in the SASYNC model.

is said to be equiangular if there exists a point p such that by drawing all n half-lines starting from p and passing through any robot, the intersections of the drawn lines with any circle centered in p form a regular n-gon. From an equiangular configuration is rather easy to accomplish the Gathering, it is enough for the robots to move toward the detected point p. From R_0, for instance, the point p is exactly $c(R)$. Hence, if the robots are always able to detect such a point during the execution of the algorithm then there is no risk to create undesired multiplicities. However, as shown in [16], the main problems start when the configuration is not equiangular but it may become as such due to some movement. This constitute the main difficulty arisen in [16] and its resolution is shown to be quite involving. It is still not clear whether by dealing with SASYNC still requires such arguments or somehow they can be simplified.

VIII. CONCLUDING REMARKS

In this work, we have studied the computational power of distributed systems consisting of very simple autonomous robots. Since it is known that the ability of a system of mobile robots to solve a given distributed problem is strongly influenced by the extent of synchrony between the robots, we have introduced a new synchronization model that is located in between ASYNC and SSYNC. This new model, called SASYNC, is defined in order to better understand the big gap between ASYNC and SSYNC in terms of level of difficulty to approach problems.

Our first result proves that SSYNC robots can solve more tasks (i.e., they are more powerful) than SASYNC robots, that in turn can solve more tasks than ASYNC robots. This is obtained by investigating the Gathering problem on rings, in particular on $\mathcal{SP}4$ configurations.

Before stating the main message that comes from this work, we need to recall the main differences between FSYNC, SSYNC, ASYNC models and the new SASYNC. One of the main difference between ASYNC and the synchronized models FSYNC and SSYNC is that in ASYNC, when a robot performs the Look phase, it can perceive other robots in each of their possible phases with respect to the LCM model. In particular, it is possible to perceive a robot while it is moving. On the contrary, in FSYNC and SSYNC, when a robot performs the Look phase, other robots are either inactive or performing the same Look phase. As an intermediate behavior, in SASYNC a robot that performs the Look phase cannot perceive other robots while these are moving, but it can perceive robots that have already computed the move but have not yet performed it. Importantly, in SASYNC a pending move will always be performed before any other move computed later in time (while in ASYNC there might occur pending moves computed very far away in time that become effective after moves computed more recently by other robots). It follows that, in SASYNC, pending moves are resolved in the order they are computed (FIFO behavior).

Concerning the obtained results and according to the discussion about the Gathering problem on the Euclidean plane, it seems that the SASYNC model is already "sufficiently hard" to deal with. In particular, it seems that studying SASYNC instead of ASYNC may already provide useful information about the complexity of the studied problem.

So, the main message coming from the new synchroniza-

tion model could be twofold: (1) the ASYNC model is hostile for designing distributed algorithms because of pending moves. Less relevance seems to concern the possibility of ASYNC robots to be seen while they are moving; (2) when a new task must be studied according to the LCM model, it is sufficient to use the SASYNC model in order to face the main difficulties that usually characterize asynchronous systems.

There are also some challenging research directions that deserve to be investigated:

- Concerning the Gathering problem on graphs, from Table 2 we see that the main question left open is whether for $n \geq 11$, configurations in $\mathcal{SP}4$ can be solved in ASYNC or not. In [30], it has been conjectured that in ASYNC such configurations are unsolvable;
- About Gathering on graphs within SASYNC, it remains open whether configurations in $\mathcal{SP}4$ with $n \geq 13$ can be solved or not;
- Another problem that remains open is whether the strict hierarchy SSYNC $>$ SASYNC $>$ ASYNC also holds for robots moving on the Euclidean plane. Moreover, for the Gathering problem, it is not clear whether for SASYNC robots it is possible to design a simpler algorithm than that provided in [16] for ASYNC robots.
- Finally, it would be interesting also to explore how the four synchronization models relate when considered within a same problem with respect to optimization issues rather than simply feasibility.

IX. ACKNOWLEDGEMENTS

The work has been supported in part by the Italian National Group for Scientific Computation GNCS-INdAM, by the Italian project "HALYomorpha halys IDentification" (HALY-ID), funded by ICT AGRIFOOD MIPAAF – CUP J99C20000820003, and by the Department of Mathematics and Computer Science, University of Perugia, Italy, project "Distributed Computing by Mobile Entities" – Fondo Ricerca di Base 2019.

REFERENCES

[1] Subhash Bhagat, Sruti Gan Chaudhuri, and Krishnendu Mukhopadhyaya. Formation of general position by asynchronous mobile robots under one-axis agreement. In Proc. 10th Int.'l WS on Algorithms and Computation (WALCOM), volume 9627 of LNCS, pages 80–91. Springer, 2016.

[2] Francois Bonnet, Maria Potop-Butucaru, and Sébastien Tixeuil. Asynchronous gathering in rings with 4 robots. In Proc. 5th Int.'l Conf. on Ad-hoc, Mobile, and Wireless Networks (ADHOC-NOW), volume 9724 of LNCS, pages 311–324. Springer, 2016.

[3] Jérémie Chalopin, Emmanuel Godard, and Antoine Naudin. Anonymous graph exploration with binoculars. In 29th International Symposium Distributed Computing (DISC), volume 9363 of LNCS, pages 107–122. Springer, 2015.

[4] Serafino Cicerone, Alessia Di Fonso, Gabriele Di Stefano, and Alfredo Navarra. Arbitrary pattern formation on infinite regular tessellation graphs. In Proc. 22nd Int.'l Conf. on Distributed Computing and Networking (ICDCN), page 56–65, New York, NY, USA, 2021. ACM.

[5] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Minimum-traveled-distance gathering of oblivious robots over given meeting-points. In Proc. 10th Int.'l Symp. on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (Algosensors), volume 8847 of LNCS, pages 57–72. Springer, 2014.

[6] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Gathering of robots on meeting-points. In Proc. 11th Int.'l Symp. on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (Algosensors), volume 9536 of LNCS, pages 183–195. Springer, 2015.

[7] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Minmax-distance gathering on given meeting-points. In Proc. 9th Int.'l Conf. on Algorithms and Complexity (CIAC), volume 9079 of LNCS, pages 127–139. Springer, 2015.

[8] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Asynchronous Pattern Formation: the effects of a rigorous approach. CoRR, abs/1706.02474, 2017.

[9] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Gathering of robots on meeting-points: feasibility and optimal resolution algorithms. Distributed Computing, 31(1):1–50, 2018.

[10] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. "Semi-Asynchronous": a new scheduler for robot based computing systems. In Proc. 38th IEEE Int.'l Conf. on Distributed Computing Systems, (ICDCS), pages 176–187. IEEE, 2018.

[11] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Asynchronous arbitrary pattern formation: the effects of a rigorous approach. Distributed Computing, 32(2):91–132, 2019.

[12] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Asynchronous robots on graphs: Gathering. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, Distributed Computing by Mobile Entities, Current Research in Moving and Computing, volume 11340 of LNCS, pages 184–217. Springer, 2019.

[13] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Gathering synchronous robots in graphs: from general properties to dense and symmetric topologies. In Proc. 26th Int.'l Colloquium on Structural Information and Communication Complexity (SIROCCO), volume 11639 of LNCS, pages 170–184. Springer, 2019.

[14] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. On gathering of semi-synchronous robots in graphs. In Proc. 21st Int,'l Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS), volume 11914 of LNCS, pages 84–98. Springer, 2019.

[15] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Gathering robots in graphs: The central role of synchronicity. Theor. Comput. Sci., 849:99–120, 2021.

[16] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. SIAM J. on Computing, 41(4):829–879, 2012.

[17] Colin Cooper, Ralf Klasing, and Tomasz Radzik. Locating and repairing faults in a network with mobile agents. Theoretical Computer Science, 411(14-15):1638–1647, 2010.

[18] Jurek Czyzowicz, Adrian Kosowski, and Andrzej Pelc. How to meet when you forget: log-space rendezvous in arbitrary graphs. Distributed Computing, 25(2):165–178, 2012.

[19] Gianlorenzo D'Angelo, Mattia D'Emidio, Shantanu Das, Alfredo Navarra, and Giuseppe Prencipe. Asynchronous silent programmable matter achieves leader election and compaction. IEEE Access, 8:207619–207634, 2020.

[20] Gianlorenzo D'Angelo, Mattia D'Emidio, Shantanu Das, Alfredo Navarra, and Giuseppe Prencipe. Leader election and compaction for asynchronous silent programmable matter. In Proc. 19th Int.'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS), pages 276–284. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

[21] Gianlorenzo D'Angelo, Gabriele Di Stefano, and Alfredo Navarra. Gathering on rings under the look-compute-move model. Distributed Computing, 27(4):255–285, 2014.

[22] Gianlorenzo D'Angelo, Alfredo Navarra, and Nicolas Nisse. A unified approach for gathering and exclusive searching on rings under weak assumptions. Distributed Computing, 30(1):17–48, 2017.

[23] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. Theor. Comput. Sci., 609:171–184, 2016.

[24] Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. Distributed Computing, 28(2):131–145, 2015.

[25] Ajoy Kumar Datta, Anissa Lamani, Lawrence L. Larmore, and Franck Petit. Ring exploration by oblivious agents with local vision. In 33rd International Conference on Distributed Computing Systems (ICDCS), pages 347–356, 2013.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3064880, IEEE Access

[26] Mattia D'Emidio, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Characterizing the computational power of mobile robots on graphs and implications for the euclidean plane. Inf. Comput., 263:57–74, 2018.

[27] Mattia D'Emidio, Daniele Frigioni, and Alfredo Navarra. Explore and repair graphs with black holes using mobile entities. Theor. Comput. Sci., 605:129–145, 2015.

[28] Mattia D'Emidio, Daniele Frigioni, and Alfredo Navarra. Characterizing the computational power of anonymous mobile robots. In Proc. 36th IEEE Int.'l Conf. on Distributed Computing Systems, (ICDCS), pages 293–302. IEEE, 2016.

[29] Stéphane Devismes, Franck Petit, and Sébastien Tixeuil. Optimal probabilistic ring exploration by semi-synchronous oblivious robots. In Proc. 16th Int.'l Colloquium on Structural Information and Communication Complexity (SIROCCO), volume 5869 of LNCS, pages 195–208, 2009.

[30] Gabriele Di Stefano, Pietro Montanari, and Alfredo Navarra. About ungatherability of oblivious and asynchronous robots on anonymous rings. In Proc. 26th Int.'l WS on Combinatorial Algorithms (IWOCA), volume 9538 of LNCS, pages 136–147. Springer, 2016.

[31] Gabriele Di Stefano and Alfredo Navarra. Gathering of oblivious robots on infinite grids with minimum traveled distance. Inf. Comput., 254:377–391, 2017.

[32] Gabriele Di Stefano and Alfredo Navarra. Optimal gathering of oblivious robots in anonymous graphs and its application on trees and rings. Distributed Computing, 30(2):75–86, 2017.

[33] Yoann Dieudonné, Franck Petit, and Vincent Villain. Leader election problem versus pattern formation problem. In Proc. 24th Int.'l Symp. on Distributed Computing (DISC), volume 6343 of LNCS, pages 267–281. Springer, 2010.

[34] Stefan Dobrev, Paola Flocchini, Rastislav Královic, and Nicola Santoro. Exploring an unknown dangerous graph using tokens. Theoretical Computer Science, 472:28–45, 2013.

[35] Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. Theor. Comput. Sci., 411(14-15):1583–1598, 2010.

[36] Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. Algorithmica, 65(3):562–583, 2013.

[37] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. Theor. Comput. Sci., 337:147–168, 2005.

[38] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. Theor. Comput. Sci., 407(1-3):412–447, 2008.

[39] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro (Eds.). Distributed Computing by Mobile Entities, Current Research in Moving and Computing, volume 11340 of LNCS. Springer, 2019.

[40] Nao Fujinaga, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. Asynchronous pattern formation by anonymous oblivious mobile robots. In Proc. 26th Int.'l Symp. on Distributed Computing (DISC), volume 7611 of LNCS, pages 312–325. Springer, 2012.

[41] Nao Fujinaga, Yukiko Yamauchi, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. Pattern formation by oblivious asynchronous mobile robots. SIAM J. Computing, 44(3):740–785, 2015.

[42] Swapnil Ghike and Krishnendu Mukhopadhyaya. A distributed algorithm for pattern formation by autonomous robots, with no agreement on coordinate compass. In Proc. 6th Int.'l Conf. on Distributed Computing and Internet Technology, (ICDCIT), volume 5966 of LNCS, pages 157–169. Springer, 2010.

[43] Taisuke Izumi, Tomoko Izumi, Sayaka Kamei, and Fukuhito Ooshita. Randomized gathering of mobile robots with local-multiplicity detection. In Proc. 11th Int.'l Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS), volume 5873 of LNCS, pages 384–398, 2009.

[44] Sayaka Kamei, Anissa Lamani, Fukuhito Ooshita, and Sebastien Tixeuil. Asynchronous mobile robot gathering from symmetric configurations without global multiplicity detection. In Proc. 37th Int.'l Symp. on Mathematical Foundations of Computer Science (MFCS), volume 7464, pages 542–553. Springer-Verlag, 2012.

[45] Ralf Klasing, Adrian Kosowski, and Alfredo Navarra. Taking advantage of symmetries: Gathering of many asynchronous oblivious robots on a ring. Theor. Comput. Sci., 411:3235–3246, 2010.

[46] Ralf Klasing, Euripides Markou, and Andrzej Pelc. Gathering asynchronous oblivious mobile robots in a ring. Theor. Comput. Sci., 390:27–39, 2008.

[47] Adrian Kosowski and Alfredo Navarra. Graph decomposition for memoryless periodic exploration. Algorithmica, 63(1-2):26–38, 2012.

[48] Adrian Kosowski, Alfredo Navarra, and Maria Cristina Pinotti. Synchronous black hole search in directed graphs. Theor. Comput. Sci., 412(41):5752–5759, 2011.

[49] Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. SIAM J. Comput., 12(4):759–776, 1983.

[50] Giuseppe Prencipe. The effect of synchronicity on the behavior of autonomous mobile robot. Theory Comput. Systems, 38:539–558, 2005.

[51] Giuseppe Prencipe. On the feasibility of gathering by autonomous mobile robots. In Proc. 12th Int.'l Colloquium on Structural Information and Communication Complexity (SIROCCO), volume 3499 of LNCS, pages 627–627, 2005.

[52] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. SIAM J. Comput., 28(4):1347–1363, 1999.

[53] Masafumi Yamashita and Ichiro Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. Theor. Comput. Sci., 411(26-28):2433–2453, 2010.

[54] Yukiko Yamauchi, Taichi Uehara, Shuji Kijima, and Masafumi Yamashita. Plane formation by synchronous mobile robots in the three-dimensional euclidean space. J. ACM, 64(3):16:1–16:43, 2017.

SERAFINO CICERONE is an Associate Professor at the Department of Information Engineering, Computer Science and Mathematics of the University of L'Aquila. He graduated in Computer Science from the University of L'Aquila in 1993 and the PhD from the University of Rome "La Sapienza" in 1998. In general, his research interests revolve around the specification, design, verification and implementation of efficient algorithms. Specific areas of interest include distributed algorithms, combinatorial optimization, algorithm engineering, algorithmic graph theory, spatial and geometric data.

GABRIELE DI STEFANO is a Full Professor at the Department of Information Engineering, Computer Science and Mathematics of the University of L'Aquila.
He received his Ph.D. at University "La Sapienza" of Rome in 1992. His current research interests include network algorithms, combinatorial optimization, algorithmic graph theory, distributed computing. He is (co-)author of more than 120 publications in journals and international conferences. He had key-participations in several EU funded projects. Among them: MILORD (AIM 2024), COLUMBUS (IST 2001-38314), AMORE (HPRN-CT-1999-00104), ARRIVAL (IST FP6-021235-2), and recently GEOSAFE (H2020-691161).

ALFREDO NAVARRA is Associate Professor since 2015 at the Mathematics and Computer Science Dept, University of Perugia, Italy. He received the PhD in Computer Science in 2004 from "Sapienza" University of Rome, Italy. Before joining the University of Perugia, he has been with various international research institutes like the INRIA of Sophia Antipolis, France; the Dept of Computer Science atthe Univ. of L'Aquila, Italy; the LaBRI, Univ. of Bordeaux, France. His research interests include algorithms, computational complexity, distributed computing and networking.