Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 322 (2016) 169–180

www.elsevier.com/locate/entcs

Synchronous Robots vs Asynchronous Lights-Enhanced Robots on Graphs[1]

Mattia D'Emidio[2]

Gran Sasso Science Institute (GSSI), Viale Francesco Crispi, I–67100, L'Aquila, Italy.
Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Via Vetoio, I–67100, L'Aquila, Italy.

Daniele Frigioni[3]

Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Via Vetoio, I–67100, L'Aquila, Italy.

Alfredo Navarra[4]

Department of Mathematics and Computer Science, University of Perugia, Via Vanvitelli, 1, I-06123, Perugia, Italy.

Abstract

In this paper, we consider the distributed setting of computational mobile entities, called robots, that have to perform tasks without global coordination. Depending on the environment as well as on the robots' capabilities, tasks might be accomplished or not.
In particular, we focus on the well-known scenario where the robots reside on the nodes of a graph and operate in Look-Compute-Move cycles. In one cycle, a robot perceives the current configuration in terms of robots positions (Look), decides whether to move toward some edge of the graph (Compute), and in the positive case it performs an instantaneous move along the computed edge (Move).
We then compare two basic models: in the first model robots are *fully synchronous*, while in the second one robots are *asynchronous* and *lights-enhanced*, that is, each robot is equipped with a constant number of lights visible to all other robots. The question whether one model is more powerful than the other in terms of computable tasks has been considered in [Das et al., *Int.'l Conf. on Distributed Computing Systems, 2012*] but for robots moving on the Euclidean plane rather than on a graph.
We provide two different tasks, and show that on graphs one task can be solved in the fully synchronous model but not in the asynchronous lights-enhanced model, while for the other task the converse holds. Hence we can assert that the fully synchronous model and the asynchronous lights-enhanced model are incomparable on graphs. This opens challenging directions in order to understand which peculiarities make the models so different.

Keywords: Distributed algorithm, Synchronicity, Mobile Robots, Luminous Robots

http://dx.doi.org/10.1016/j.entcs.2016.03.012

1 Introduction

In the last few years a considerably large amount of research, in the area of distributed computing, has been devoted to the study of models and algorithmic approaches for the so-called *robot-based computing systems*, due to their importance in a wide range of real-world applications. In this kind of systems a set of mobile entities, usually referred as *robots*, have either to perform tasks and/or to achieve goals under a variety of assumptions that depend on the considered scenario [6,14].

Particular efforts have been dedicated to models where robots are *autonomous*, i.e. they act without a central control, and operate in a *Look-Compute-Move* (LCM) operational mode (see [1,2,3,19,22] and references therein). In such a model, which has become a prominent one in the area of distributed algorithms for robot-based computing systems, robots operate in so-called *LCM cycles*. During each cycle, a robot acquires a snapshot of the surrounding environment (*Look* phase), then executes an appropriate algorithm, which is the same for all robots, by using the obtained snapshot as input (*Compute* phase), and finally moves toward a desired destination, if any (*Move* phase).

Several modeling assumptions have been also considered that can affect the computational power of the robots. In particular, in some cases, robots may have distinct identities, i.e. each robot is associated with a different identifier that can be used during the Compute phase. If robots are without identifiers, they are said to be *anonymous*. In some other cases, robots may have a finite but *persistent* memory device whose content is preserved from one LCM cycle to the next; robots are said to be *oblivious* if they do not, which means that they start each cycle without any information about what happened in the past.

In this research area, many different problems and tasks have been studied: robots might be asked to gather in certain specific locations [20] (also known as the *Gathering* problem), to form a desired geometric pattern [21] (also known as the *Pattern Formation* problem), or to explore an unknown area [17] (also known as the *Exploration* problem). In addition, several different settings have been investigated. Robots can move on a Euclidean plane [16], or they are constrained to move on a given input graph, which can either be known in advance [9] or not [4]. Robots can be able to communicate, e.g. by means of tokens as in [15], or not [18]. Finally, there might exist or not an objective function to be optimized, associated with the problem (see [3,11,12,13] and references there in). For instance, one may ask for the minimum number robots, the minimum number of steps performed by all the robots, or the minimization of the maximum number of steps performed by a single robot, to achieve a certain goal.

Look-Compute-Move cycles might be subject to different temporal constraints

[1] The work has been partially supported by the Italian Ministry of Education, University, and Research (MIUR) under national research projects: PRIN 2010N5K7EB "ARS TechnoMedia – Algoritmica per le Reti Sociali Tecno-Mediate" and by the National Group for Scientific Computation (GNCS-INdAM).

[2] Email: mattia.demidio@univaq.it

[3] Email: daniele.frigioni@univaq.it

[4] Email: alfredo.navarra@unipg.it

dictated by the considered schedule. The most common models in the literature are the following:

Fully-synchronous ($\mathcal{FSYNC}$) [8]: The activation phase of all robots can be logically divided into global rounds, where in each round the robots are all activated, obtain the same snapshot of the environment, compute and perform their move. Notice that, this assumption is computationally equivalent to a fully synchronized system in which robots are activated simultaneously and all operations happen instantaneously.

Semi-synchronous ($\mathcal{SSYNC}$) [10]: It coincides with the $\mathcal{FSYNC}$ model with the only difference that not all robots are necessarily activated in each round.

Asynchronous ($\mathcal{ASYNC}$) [5]: The robots are activated independently, and the duration of each *Compute*, *Move* and inactivity phase is finite but unpredictable. 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.

Recently, further models have been introduced by Das at al. in [6], extending the aforementioned ones. In detail, given a model $\mathcal{M} \in \{\mathcal{FSYNC}, \mathcal{SSYNC}, \mathcal{ASYNC}\}$, the authors define models $\mathcal{M}^k$, where each robots operating in the $\mathcal{M}$ model is equipped with a *light* that is visible to itself and to the other robots during the *Look* phase. The light associated with a robot can assume k different colors and can be updated by a robot during its *Compute* phase. The light is assumed to be *persistent*, i.e. despite robots can be oblivious, their lights are not automatically reset at the end of a cycle. Thus, robots' lights can be considered as a form of external persistent memory. Light-enhanced robots are usually referred as *luminous robots*. Note that, depending on the considered scenario, a robot might have visibility of the lights of either all other robots of just of a subset of them. Another model, introduced in [6], considers oblivious robots which are "slightly" empowered with the additional capability of remembering a constant number of previously acquired snapshots. More precisely, for some integer constant $j > 0$, each robot is allowed to store in an internal memory at most j snapshots from previous Look phases. When robots, operating in a certain model $\mathcal{M}$, are endowed with the ability of remembering k snapshots, we refer to the modified model as $\mathcal{M}_k$ and to the robots as *partially oblivious* robots.

1.1 Motivation of the paper

Our work is inspired by the paper of Das et al. [6], which concerned with the problem of comparing the computational capabilities of robots operating in LCM model and moving in the Euclidean plane, under different levels of synchronization, with those of both *luminous* robots and *partially oblivious* robots. In details, the authors of [6] provided a series of results that prove relationships of dominance between classic models and variations of them that consider the capability of either using lights or snapshots, or a combination of them.

In Figure 1, we summarize the main contributions of [6] and show such relation-

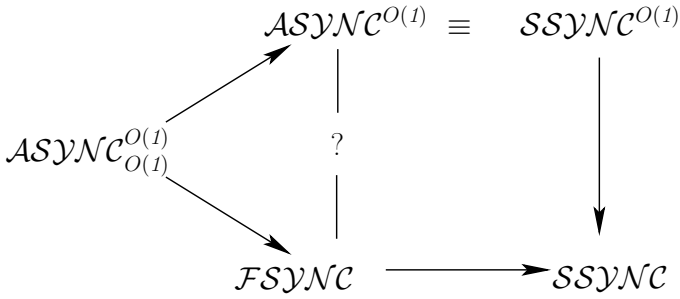

Figure 1. Relationships among models.

ships of dominance. We denote by a direct arrow between two models, say A and B the fact that every problem solvable in A is also solvable in B. Moreover, we denote by a triple bar symbol the fact that every problem solvable in A is also solvable in B and viceversa. In particular, the authors of [6] showed that:

- $\mathcal{ASYNC}^{O(1)} \equiv \mathcal{SSYNC}^{O(1)} \rightarrow \mathcal{SSYNC}$: robots that operate in $\mathcal{ASYNC}$ and are endowed with a constant number of lights (i.e. that operate in $\mathcal{ASYNC}^{O(1)}$) are computationally as powerful as robots that operate in $\mathcal{SSYNC}$ and are endowed with a constant number of lights (i.e. that operate in $\mathcal{SSYNC}^{O(1)}$); moreover, both $\mathcal{ASYNC}^{O(1)}$ and $\mathcal{SSYNC}^{O(1)}$ models are computationally more powerful than $\mathcal{SSYNC}$.
- $\mathcal{ASYNC}^{O(1)}_{O(1)} \rightarrow \mathcal{ASYNC}^{O(1)}$: robots that operate in $\mathcal{ASYNC}$ and are endowed with both a constant number of lights and the ability to remember a constant number of snapshots (i.e. that operate in $\mathcal{ASYNC}^{O(1)}_{O(1)}$) are more powerful than robots that operates in $\mathcal{ASYNC}^{O(1)}$. Moreover, $\mathcal{ASYNC}^{O(1)}_{O(1)}$ is more powerful than $\mathcal{FSYNC}$.

In Figure 1 we also report relationships of dominance that were already well known from the literature, like e.g. $\mathcal{FSYNC} \rightarrow \mathcal{SSYNC}$. Note that, in [6] the relationship between $\mathcal{FSYNC}$ and $\mathcal{ASYNC}^{O(1)}$ has not been established.

1.2 Contribution of the paper

In this paper, we try to answer to the problem of finding the relationship, in terms of computable tasks, that exists between $\mathcal{FSYNC}$ and $\mathcal{ASYNC}^{O(1)}$. To this regard, we provide two different distributed tasks of robots moving within a graph environment, and not in the Euclidean plane as in [6], and show that one task can be solved in $\mathcal{FSYNC}$ but not in $\mathcal{ASYNC}^{O(1)}$, while for the other the viceversa holds. Hence, we provide a proof that $\mathcal{FSYNC}$ and $\mathcal{ASYNC}^{O(1)}$ are incomparable on graphs. This opens challenging directions in order to understand which peculiarities make the models so different.

1.3 Structure of the paper

In Section 2, we describe the distributed system in which the robots operate. In Section 3, we define a distributed problem which can be solved in $\mathcal{FSYNC}$ but not

in $\mathcal{ASYNC}^{O(1)}$. In Section 4, we define a distributed problem which can be solved in $\mathcal{ASYNC}^{O(1)}$ but not in $\mathcal{FSYNC}$. Finally, in Section 5, we conclude the paper and provide some possible future research directions.

2 Preliminaries

In this paper, as already mentioned, we consider a system composed of a team of mobile entities, called *robots*, that operates in Look-Compute-Move cycles. In particular, each robot is modeled as an independent computational unit with its own local memory and capable of performing local computations. The robots are placed in a spatial environment, which can be assumed to be an undirected graph $G = (V, E)$, i.e. robots are placed on the nodes of the graph. Therefore, each robot has its own local perception of the surrounding environment, which means it can detect a graph isomorphic to G and understand whether a node is occupied by a robot or not. Each robot is equipped with sensing capabilities that return a *snapshot* of the relative positions of all other robots with respect to its location on the perceived graph.

In the remaining of the paper, we assume that robots are *anonymous* and *identical*, i.e. they are indistinguishable by their appearance, and execute the same algorithm. They are *oblivious*, i.e. they have no memory. Moreover, we consider that robots act without a central control, i.e. they are assumed to be *autonomous* and are not able to directly communicate information (e.g. by a wireless interface) with other robots, i.e. they are *silent*. Each robot is endowed with motor capabilities and can freely move on G. However, the movement along one edge of G is considered instantaneous, so that each time a robot perceives the snapshot of the current configuration, it sees all other robots always on the nodes of G. We will specify different assumptions when it is not clear from the context.

At any point in time, a robot is either *active* or *inactive*. When active, a robot executes a *Look-Compute-Move* (LCM) cycle performing the following three operations in sequence, each of them associated with a different state:

Look: The robot observes the world by activating its sensors, which return a snapshot of the positions of all robots with respect to its own perception.

Compute: The robot executes its algorithm, using the data sensed in the Look phase as input. The result of this phase is a target (destination) node.

Move: The robot moves toward the computed target: if the destination is the current position, the robot simply stays still, i.e. it performs what we call a *null* movement.

When inactive, instead, a robot is idle. All robots are initially inactive. The amount of time to complete a full LCM cycle is assumed to be finite but unpredictable.

3 $\mathcal{ASYNC}^{O(1)}$ is not more powerful than $\mathcal{FSYNC}$

In this section, we define a distributed problem, namely the *Pattern Sequence Chasing* (PSC) problem, and show that it can be solved in $\mathcal{FSYNC}$ but not in $\mathcal{ASYNC}^{O(1)}$. The PSC problem can be thought as a variation of the *Series of Patterns Formation* problem, defined in [7], where: i) no pattern is repeated in the provided sequence; ii) robots move on a graph instead of a Euclidean plane.

The problem is defined as follows: as an input, we consider a set of k robots disposed on a *non-anonymous* undirected and complete graph G (i.e. each node of G is associated with a unique identifier), and an array A of dimension n whose entries are pairwise distinct patterns. A pattern P is an ordered array of k distinct identifiers of nodes of G that represents a placement of the k robots on the nodes identified by P. Initially, the robots are disposed according to pattern $A[0]$. Subsequently, the k robots have to move in order to form pattern $A[1]$, and so forth, moving from pattern $A[i]$ to pattern $A[(i+1) \mod n]$, see for instance Figure 2.

Note that, since the robots are identical and anonymous, assuming that the nodes of G can be uniquely identified is crucial to the feasibility of the problem. In fact, in anonymous graphs, if two robots occupy the same node, then the adversary can make both robots always move concurrently. In this way they will never reach different destinations, hence the problem becomes unsolvable.

For the sake of simplicity, we consider that A has finite size n. Therefore, without loss of generality, robots are asked to move from a pattern $A[i \mod n]$ to a pattern $A[(i+1) \mod n]$.

In the remaining of the paper, we denote by $A[i][j]$ the j-th entry of pattern $A[i]$.

Pattern Series Chasing (PSC) Problem

***Input*:** A non-anonymous undirected and complete graph G. An array A of n patterns, each involving k nodes of G, such that $A[i] \neq A[j]$, for every $0 \leq i \neq j < n$. A set of k robots forming $A[0]$ in G.

***Solution*:** A distributed algorithm that ensures robots to form pattern $A[(i+1) \mod n]$ after $A[i \mod n]$, $i \in \mathbb{N}$.

In what follows, we provide an algorithm, namely Algorithm PatternChaser (see Algorithm 1), which solves PSC in $\mathcal{FSYNC}$ by exploiting the uniqueness of the current pattern in the sequence defined by the entries of the array A. The algorithm works as follows: during each Look phase, each robot perceives a snapshot s of the positions (i.e., ids of nodes of the graph) of other robots. Such snapshot corresponds to a certain pattern $A[i]$ in the sequence which can be easily found by scanning A (initially the snapshot is clearly equal to $A[0]$). Then, given $A[i]$, each robot compares the id of the node of the graph he is located at, say v, with those in $A[i]$ by sequentially scanning it. Eventually, each robot finds the index j such that $A[i][j] = v$, and can perform its move from position $A[i][j]$ to position $A[(i+1) \mod n][j]$.

Algorithm 1 Algorithm for solving PSC under $\mathcal{FSYNC}$.

```
1: procedure PatternChaser(Snapshot s)   ▷ Snapshot s acquired during the Look phase
2:     i := 0;
3:     while A[i] ≠ s do
4:         i++;
5:     end while
6:     Let my_p be my position within pattern A[i];
7:     j := 0;
8:     while my_p ≠ A[i][j] do
9:         j++;
10:    end while
11:    The new position is A[(i + 1) mod n][j];
12: end procedure
```

Theorem 3.1 *Algorithm* PatternChaser *correctly solves PSC problem in* $\mathcal{FSYNC}$.

Proof As robots are fully synchronous, every time a new Look-Compute-Move cycle starts, all robots perceive the same snapshot s (at the beginning, s is equal to $A[0]$). Therefore, during each Compute phase, each robot can: i) uniquely find the current pattern in the sequence A, say $A[i]$; ii) find the index j within the array $A[i]$ such that $A[i][j]$ matches its position. Since all such entries are distinct, each of them uniquely defines the next position at which robots should be placed in the subsequent step, namely the j-th entry of array $A[(i+1) \mod n]$, i.e. $A[(i+1) \mod n][j]$. The movement is accomplished by letting all robots move simultaneously to reach the desired placement. □

On the contrary, in general, operating under $\mathcal{ASYNC}^{O(1)}$ does not permit to solve PSC, unless the number of provided lights is $O(n)$, which is not necessarily a constant number.

Theorem 3.2 *Problem PSC cannot be solved in* $\mathcal{ASYNC}^{O(1)}$.

Proof To prove the statement, we have to show that there is at least one case where a constant number of lights does not suffice to robots to infer the current pattern $A[i]$, and therefore to move to the correct subsequent pattern $A[(i+1) \mod n]$. In particular, we prove that any algorithm requires robots to be equipped with a number of lights that is not constant with respect to the size n of the input array A, thus making the problem unsolvable in $\mathcal{ASYNC}^{O(1)}$. In fact, if the number of lights permits to encode the current index i, then the robots may synchronize themselves to obtain $A[(i+1) \mod n]$ from $A[i]$.

To prove the statement true for the general case of k robots, we make use of an example that can be easily extended to higher dimensions. As input we consider a set of $k = 3$ robots, a non-anonymous complete graph $G = (V, E)$ with $V = \{v_1, \cdots, v_7\}$, and a sequence of patterns A (part of which is shown in Figure 2). We

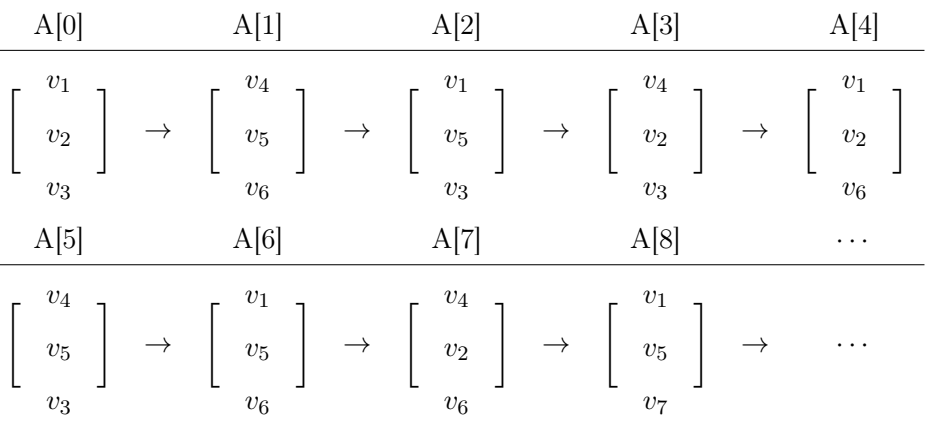

Figure 2. Example of an instance that is not solvable in $\mathcal{ASYNC}^{O(1)}$ by $k = 3$ luminous robots.

now show that such an instance of PSC cannot be solved by asynchronous robots equipped with a constant number of lights.

Let us assume that, at step $i = 0$, the 3 robots are correctly placed on the input graph G, i.e. each one of them is located exactly at one of the nodes of the required pattern, say $A[0] = \{v_1, v_2, v_3\}$, as shown in Figure 2.

From $A[0]$ robots have to move to $A[1]$. Lights do not permit to make all robots move concurrently, and by hypothesis lights do not suffice to encode index i. Three cases may arise concerning the movement of robots from $A[0]$: (i) all robots move; (ii) only one robot moves; (iii) only two robots move. In case (i), configuration $A[1]$ is correctly reached. In case (ii), configurations $A[2]$, $A[3]$ or $A[4]$ can be reached. In case (iii), configurations $A[5]$, $A[6]$ or $A[7]$ can be reached.

It follows that, if robots cannot encode index i, then in some cases they can "get confused" about which pattern must be realized. Lights might be used to infer whether all robots have moved or not. However, for instance, from pattern $A[6]$ it is not sufficient for the third robot to know that the other two robots have moved in order to understand whether the previous pattern was $A[1]$ or $A[7]$.

A possible approach to overcome this limitation is that of using moves to intermediate patterns, that could help to infer the target pattern. However, also this approach is not effective. In fact, since there are no restrictions on the size n of A, we may consider A composed of all possible patterns, and therefore each intermediate pattern coincides with an entry of A. Again, robots can "get confused" about which pattern must be realized. □

As shown in the proof of the above theorem, problem PSC cannot be solved in $\mathcal{ASYNC}^{O(1)}$ because there is no mean to encode the current configuration by exploiting a constant number of lights nor some strategic positioning of the robots. However, if we assume robots empowered also with a constant persistent memory able to store one snapshot, then the index of the current pattern is deducible from the snapshot, and hence the following corollary can be stated.

Corollary 3.3 *Problem PSC can be solved in $\mathcal{ASYNC}^{O(1)}_{O(1)}$.*

Similarly to the above corollary, instead of adding persistent memory, if we increase the number of lights to be enough to encode the index of the current pattern, then the following corollary can be stated.

Corollary 3.4 *Problem PSC can be solved in* $\mathcal{ASYNC}^{O(\log n)}$.

4 $\mathcal{FSYNC}$ is not more powerful than $\mathcal{ASYNC}^{O(1)}$

In this section, we define a distributed problem, namely the *Forth and Back* (FB) problem, and show that it can be solved in $\mathcal{ASYNC}^{O(1)}$ but not in $\mathcal{FSYNC}$.

Forth and Back (FB) Problem

***Input*:** Two robots, named r_1 and r_2, and a graph G, which is a path P of at least six nodes. The two robots reside at distinct nodes of P, which are neither the endpoints of P nor neighbors of the endpoints of P. Let d be the initial distance, i.e number of edges, between the two robots.

***Solution*:** A distributed algorithm that ensures robots to alternate their distance between $d+2$ and d.

The FB problem has some similarities with our previous PSC problem, but the sequence of patterns (distances) is not defined according to the ids of the nodes but on the initial configuration where robots always start lying in distinct internal nodes. Moreover, notice that, in this case, the graph is assumed to be anonymous.

Theorem 4.1 *Problem FB cannot be solved in* $\mathcal{FSYNC}$.

Proof Whenever a Look-Compute-Move cycle starts, robots cannot infer whether the current distance must be enlarged or restricted. In fact, this information requires to know either how many times robots have executed their running cycles, or at least whether such a number is odd or even. These information simply cannot be deduced from the snapshot acquired during the Look phase. □

In what follows we provide an algorithm, namely Algorithm ForthBack (see Algorithm 2), which assumes that robots are empowered with two different lights, each of them assuming two different colors. Algorithm 2 makes use of colors white, black for light 1, and red, blue for light 2, with the following meanings:

- white: indicates that the robot is ready to move for the next step;
- black: indicates that the robot has moved and it is ready for the next step;
- red: indicates an even step where the previous distance must be increased;
- blue: indicates an odd step where the previous distance must be decreased.

At the beginning, both robots start with the lights set to white and red. As we will see, algorithm ForthBack ensures that, whenever both robots have the first light set to white, they also have the second light concordant, which means they are currently synchronized. That is, Algorithm 2 solves FB in $\mathcal{ASYNC}^{O(1)}$ by

exploiting the parity encoding of the current step by means of lights. Of course, the same result can be obtained by making use of only one light assuming four different colors.

Algorithm 2 Algorithm for solving FB under $\mathcal{ASYNC}^{O(1)}$.

```
1: procedure FORTHBACK(Colors l'_1 and l'_2) ▷ Lights of the other robot acquired during the Look phase
2:     Let l_1 and l_2 the current colors of my two lights;
3:     Let d be the number of edges from the other robot;
4:     if l_1 = WHITE ∧ l_2 = RED ∧ l'_2 = RED then
5:         l_1 := BLACK;
6:         Let v be the neighbor at distance d + 1 from the other robot;
7:         The new position is v;
8:         Exit;
9:     end if
10:    if l_1 = WHITE ∧ l_2 = BLUE ∧ l'_2 = BLUE then
11:        l_1 := BLACK;
12:        Let v be the neighbor at distance d − 1 from the other robot;
13:        The new position is v;
14:        Exit;
15:    end if
16:    if l_1 = BLACK ∧ l_2 = RED ∧ ((l'_1 = WHITE ∧ l'_2 = BLUE) ∨ (l'_1 = BLACK ∧ l'_2 = RED)) then
17:        l_1 := WHITE;
18:        l_2 := BLUE;
19:        Exit;
20:    end if
21:    if l_1 = BLACK ∧ l_2 = BLUE ∧ ((l'_1 = WHITE ∧ l'_2 = RED) ∨ (l'_1 = BLACK ∧ l'_2 = BLUE)) then
22:        l_1 := WHITE;
23:        l_2 := RED;
24:        Exit;
25:    end if
26: end procedure
```

Theorem 4.2 *Algorithm* FORTHBACK *correctly solves* FB *problem in* $\mathcal{ASYNC}^{O(1)}$.

Proof Let us denote as l_1, l_2 (l'_1, l'_2, respectively) the lights of r_1 (r_2, respectively).

If l_1 is WHITE, then r_1 is ready to accomplish a movement, which either must enlarge the distance of the previous placement or restrict it. This depends on l_2, whether it is RED or BLUE, respectively. Moreover, in order to understand the right movement, r_1 considers also the two lights of r_2, namely l'_1 and l'_2. If they are the same, i.e. $l_1 = l'_1$ and $l_2 = l'_2$, then this is the case where robots are currently synchronized. If $l_1 = l'_1 =$BLACK and l_2 is discordant with respect to l'_2, then r_1

waits for r_2 in order to get synchronized with it. If l'_1 is BLACK and $l_2 = l'_2$, then r_1 concludes that r_2 has already moved, so it is now its turn. The movement to be done is evaluated according to the current lights of r_2. In details, if l_2 is RED, then r_1 must move away from r_2 of one edge. In fact, as we noticed, r_2 has already terminated its movement. Hence, it is waiting for r_1. If d was the distance between the two robots before both robots moved, then $d+2$ must define the final placement of the two robots in the current step. Since they both contribute of one edge, when only one robot moves, the current distance d' is $d' = d + 1$, Therefore, by moving of one edge, the two robots will reach distance $d + 2$. Similarly, If l_2 is BLUE, then r_1 must move closer to r_2 of one edge.

Finally, if l_1 is BLACK, then r_1 has terminated a step and either it must start the next one, or it has to wait for r_2 in order to temporarily get synchronous with it. This is realized by considering the lights of r_2. If l'_1 is WHITE and $l_2 \neq l'_2$, then r_1 changes l_1 to WHITE and l_2 to l'_2. If l'_1 is WHITE and $l_2 = l'_2$, then r_1 has to wait for r_2 to move. If l'_1 is BLACK and $l_2 = l'_2$, then r_1 changes l_1 to WHITE and makes l_2 discordant with l'_2. The case where l'_1 is BLACK and $l_2 \neq l'_2$ cannot occur. This realizes the synchronization among robots. □

5 Conclusion

We have shown the incomparability of two important models studied in robot-based computing systems. In particular, we have considered $\mathcal{FSYNC}$, where all robots execute their algorithm synchronously, and $\mathcal{ASYNC}^{O(1)}$, where robots are asynchronous but they are empowered with a constant number of lights. The proof has been pursued by providing two different distributed tasks of robots moving within a graph environment, and show that one task can be solved in $\mathcal{FSYNC}$ but not in $\mathcal{ASYNC}^{O(1)}$, while for the other the viceversa holds.

The case where robots move on the Euclidean plane remains open. Actually, arguments of Theorems 3.1, 4.1 and 4.2 can be extended quite easily in order to hold in the Euclidean case as well. The main challenge remains Theorem 3.2 where the difficulty mainly resides in proving that stigmergy strategies cannot be effective. That is, in the Euclidean plane, robots may compute intermediate positions rather than straightly move toward the position dictated by the current pattern. This opens challenging directions in order to understand which peculiarities make the $\mathcal{FSYNC}$ and the $\mathcal{ASYNC}^{O(1)}$ models so different.

Another interesting direction is that of revisiting the whole picture of Figure 1 when robots move on a graph environment.

References

[1] Chatterjee, A., S. G. Chaudhuri and K. Mukhopadhyaya, *Gathering asynchronous swarm robots under nonuniform limited visibility*, in: *11th International Conference on Distributed Computing and Internet Technology (ICDCIT)*, Lecture Notes in Computer Science **8956** (2015), pp. 174–180.

[2] Cicerone, S., G. Di Stefano and A. Navarra, *Minimum-traveled-distance gathering of oblivious robots over given meeting points*, in: *10th International Symposium on Algorithms and Experiments for Sensor*

Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS), Lecture Notes in Computer Science **8847** (2014), pp. 57–72.

[3] Cicerone, S., G. Di Stefano and A. Navarra, *Minmax-distance gathering on given meeting points*, in: *9th International Conference on Algorithms and Complexity (CIAC)*, Lecture Notes in Computer Science **9079** (2015), pp. 127–139.

[4] Czyzowicz, J., A. Kosowski and A. Pelc, *How to meet when you forget: log-space rendezvous in arbitrary graphs*, Distributed Computing **25** (2012), pp. 165–178.

[5] D'Angelo, G., G. Di Stefano and A. Navarra, *Gathering on rings under the look-compute-move model*, Distributed Computing **27** (2014), pp. 255–285.

[6] Das, S., P. Flocchini, G. Prencipe, N. Santoro and M. Yamashita, *The power of lights: Synchronizing asynchronous robots using visible bits*, in: *32nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2012, pp. 506–515.

[7] Das, S., P. Flocchini, N. Santoro and M. Yamashita, *On the computational power of oblivious robots: Forming a series of geometric patterns*, in: *29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)* (2010), pp. 267–276.

[8] Degener, B., B. Kempkes, T. Langner, F. Meyer auf der Heide, P. Pietrzyk and R. Wattenhofer, *A tight runtime bound for synchronous gathering of autonomous robots with limited visibility*, in: *23rd ACM Symp. on Parallelism in algorithms and architectures (SPAA)*, 2011, pp. 139–148.

[9] D'Emidio, M., D. Frigioni and A. Navarra, *Exploring and making safe dangerous networks using mobile entities*, in: *12th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW)*, Lecture Notes in Computer Science **7960** (2013), pp. 136–147.

[10] Devismes, S., F. Petit and S. Tixeuil, *Optimal probabilistic ring exploration by semi-synchronous oblivious robots*, in: *16th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Lecture Notes in Computer Science **5869**, 2009, pp. 195–208.

[11] Di Stefano, G. and A. Navarra, *Gathering of oblivious robots on infinite grids with minimum traveled distance*, Information and Computation. To appear.

[12] Di Stefano, G. and A. Navarra, *Optimal gathering of oblivious robots in anonymous graphs*, in: *Proceedings of the 20th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Lecture Notes in Computer Science **8179**, 2013, pp. 213–224.

[13] Di Stefano, G. and A. Navarra, *Optimal gathering on infinite grids*, in: *Proc. of the 16th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, Lecture Notes in Computer Science **8756**, 2014, pp. 211–225.

[14] Dieudonné, Y., A. Pelc and D. Peleg, *Gathering despite mischief*, ACM Transactions on Algorithms **11** (2014), p. 1.

[15] Dobrev, S., P. Flocchini, R. Královic and N. Santoro, *Exploring an unknown dangerous graph using tokens*, Theoretical Computer Science **472** (2013), pp. 28–45.

[16] Flocchini, P., D. Ilcinkas, A. Pelc and N. Santoro, *Remembering without memory: Tree exploration by asynchronous oblivious robots*, Theoretical Computer Science **411** (2010), pp. 1583–1598.

[17] Flocchini, P., D. Ilcinkas, A. Pelc and N. Santoro, *Computing without communicating: Ring exploration by asynchronous oblivious robots*, Algorithmica **65** (2013), pp. 562–583.

[18] Flocchini, P., D. Ilcinkas, A. Pelc and N. Santoro, *Computing without communicating: Ring exploration by asynchronous oblivious robots*, Algorithmica **65** (2013), pp. 562–583.

[19] Flocchini, P., G. Prencipe and N. Santoro, *Distributed computing by oblivious mobile robots*, Synthesis Lectures on Distributed Computing Theory **3** (2012), pp. 1–185.

[20] Flocchini, P., G. Prencipe, N. Santoro and P. Widmayer, *Gathering of asynchronous robots with limited visibility*, Theoretical Computer Science **337** (2005), pp. 147–168.

[21] Fujinaga, N., Y. Yamauchi, S. Kijima and M. Yamashita, *Asynchronous pattern formation by anonymous oblivious mobile robots*, in: *Distributed Computing*, Lecture Notes in Computer Science **7611**, Springer, 2012 pp. 312–325.

[22] Kamei, S., A. Lamani, F. Ooshita and S. Tixeuil, *Gathering an even number of robots in an odd ring without global multiplicity detection*, in: *Mathematical Foundations of Computer Science (MFCS)*, Springer, 2012 pp. 542–553.