PAIRBOT: A NOVEL MODEL FOR AUTONOMOUS MOBILE ROBOT SYSTEMS CONSISTING OF PAIRED ROBOTS

Yonghwan Kim
Department of Computer Science
Nagoya Institute of Technology
Nagoya, Japan
kim@nitech.ac.jp

Yoshiaki Katayama
Department of Computer Science
Nagoya Institute of Technology
Nagoya, Japan
katayama@nitech.ac.jp

Koichi Wada
Department of Applied Informatics
Hosei University
Tokyo, Japan
wada@hosei.ac.jp

ABSTRACT

Programmable matter (PM) is a form of matter capable of dynamically altering its physical properties, such as shape or density, through programmable means. From a robotics perspective, PM can be realized as a distributed system consisting of numerous small computational entities working collaboratively to achieve specific objectives. Although autonomous mobile robot systems serve as an important example and have been researched for more than two decades, these robots often fail to perform even basic tasks, revealing a considerable gap in PM implementation.

In this paper, we introduce a novel computational paradigm, termed the Pairing Robot model (Pairbot model), which is built on an autonomous mobile robot system. In this model, each robot forms a pair with another, enabling them to recognize each other and adapt their positions to achieve designated goals. This fundamental principle of pairing substantially enhances inter-robot connectivity compared to conventional LCM-type model, even under asynchronous scheduler conditions. This shift has considerable implications for computational capabilities, specifically in problem solvability.

We explore two specific challenges—the perpetual marching problem and the 7-pairbots-gathering problem—to demonstrate the computational power of Pairbot model. This model provides new avenues and insights to address inherent issues in autonomous mobile robots.

Keywords pairbot · programmable matter · autonomous mobile robots · LCM model

1 Introduction

1.1 Background

The foundational concept of a distributed system composed of multiple robots was initially introduced in a landmark paper [Suzuki and Yamashita, 1999]. Within this framework, each robot autonomously observes the positions of the other robots and moves to a new position based on a prescribed algorithm. These robots are anonymous (i.e., indistinguishable in appearance) and uniform (i.e., executing identical algorithms). In the work cited [Suzuki and Yamashita, 1999], this conceptual model for mobile robots is designated as the LCM (Look-Compute-Move) model. For simplicity, we will refer to robots operating under the conventional LCM model without specific assumptions such as light (Das et al., 2015) as LCM-robots. The paper provides a comprehensive analysis of the capabilities and constraints of this distributed system, exploring issues such as pattern formation and agreement problems. Since the introduction of the LCM model, much related work has been studied to clarify its computational power and limitations [Flocchini et al., 2019], [Prencipe, 2014] for more than 20 years.

Numerous studies have focused primarily on the relationship between the computational capabilities of each robot and the solvability of the given problem, such as problems of gathering [Cicerone et al., 2018], [Cieliebak et al., 2012],...
1.2 Related Work

Building on the seminal work by Suzuki and Yamashita [1999], which introduced the autonomous mobile robot model and explored its computational capabilities, extensive research has since been investigated concerning its computational power and limitations of these robots in various distributed coordination tasks, such as rendezvous [Izumi et al. 2011], gathering [Cicerone et al. 2018], [Cieliebak et al. 2012], pattern formation [Flocchini et al. 2008, 1999], [Fujinaga et al. 2015], dispersion [Kshemkalyani et al. 2020], [Molla et al. 2021], [Augustine and Moses 2018], Barriere et al. [2009], and flocking [Gervasi and Prencipe 2004], [Soussi et al. 2009].

In the early years of the study, robots are modeled as points (i.e., without volume). However, many recent studies consider robots that have volume, known as fat robots [Czyzowicz et al. 2009], or opaque robots (each robot may obstruct the view of the other robots): these models present new paradigms for problems such as the complete visibility problem [Aljohani et al. 2018], [Kim et al. 2023]. Many studies aim to clarify the relationship between robots' capabilities and the solvability of various problems [Flocchini et al. 2019], [Prencipe 2014], [Buchin et al. 2021].

Programmable matter (PM), which is defined as a matter that can change its physical properties, such as shape, color, and stiffness, in response to a computer program, was first introduced in [Toffoli and Margolus 1993], [Goldstein et al. 2005]. Realizing programmable matter could bring several potential advantages, including: (1) flexibility: PM makes matters highly flexible and adaptable, which allows PM to be used in a variety of applications such as wearable devices and soft robotics [Bongard 2013], [Kim et al. 2013]. (2) customization: PM could allow customization of objects and devices to meet specific needs and requirements, e.g., tailored clothing or customized prosthetics, and (3) interactivity: PM could bring new levels of interactivity and responsiveness to physical objects, e.g., objects that respond to touch, sound, or light in creative ways. These potential benefits of PM are significant and could lead to many new and innovative applications. However, PM is a complex and challenging task that requires solving several technical and scientific problems, and thus it is still in the research stage in various fields such as robotics, computer science, and material science. PM can be implemented through the use of small robots, such as nanorobots [Cerofolini et al. 2010], [Yarin 2010], that can work together to form larger structures. Many studies on autonomous mobile robot systems have the potential to be useful in realizing PM. However, autonomous mobile robots require various additional capabilities in many cases, and some of them are not necessarily realistic, as they have many difficulties, even in simple problems.

As a new computational model for the realization of programming matter, a self-organizing particle system, called Amoebot model, was first introduced in [Derakhshandeh et al. 2014]. The Amoebot model consists of a large number of computational particles placed on a triangular grid plane, which locally interact with each other to solve problems. Each particle repeatedly changes its state to contraction or expansion, which means the state by which a particle occupies one node and two adjacent nodes. The Amoebot model allows for coordinated movement between two connected particles, called handover. A handover is an interaction in which a particle can contract out of a certain node at the same time that another particle expands into that node. The Amoebot model can solve various distributed coordination problems, such as the universal coating problem [Daymude et al. 2017a], leader election [Bazzi and Briones 2019], [Daymude et al. 2017b], and convex hull formation [Daymude et al. 2020]. Furthermore, recent work has introduced the canonical amoebot model, which considers concurrent control [Daymude et al. 2021].

Silbot model [D’Angelo et al. 2020] is another computational model for PM that introduces a new modeling approach by relaxing some of the assumptions made in the previous model. In the Silbot model, each particle cannot communicate with the others (i.e., silent) and operates asynchronously (i.e., in full synchronicity). However, this model requires a specific symmetry-breaking capability; if two or more particles attempt to expand toward the same cell, only one of them succeeds. The literature [D’Angelo et al. 2020] shows that a leader election can be achieved from a connected configuration with certain restrictions (or with some additional assumptions).

MObLOT [Cicerone et al. 2021] is a computational model designed from a different point of view, which extends the oblivious mobile robot model to address a wider spectrum of cases. The main difference of the MOBLOT model is heterogeneous system; all (oblivious) robots are partitioned into 2 types, colored black or white. This feature enables simple (oblivious) robots to perform more complex tasks.
The literature [Daymude et al. 2021, D’Angelo et al. 2020, Cicerone et al. 2021] shows that these new computational models for PM have great computational power in some problems; however, they require some different assumptions which are not considered (or cannot be used) in the conventional autonomous mobile robot model, such as explicit communication or handover. Here, we expect that if we can design a new computational model as close as possible to the conventional LCM model [Suzuki and Yamashita 1999], we can use many results among many existing results investigated for a long time.

1.3 Comparison with the Other Computational Models

Here we provide a comparison between our proposed model Pairbot model and three other computational models: Amoebot, SILBOT, and MOBLOT. It is important to note that these models solve different problems based on different assumptions; thus, it is difficult to provide a direct comparison of their computational power.

The Amoebot model is one of the most popular PM models. Various Amoebot models presented in many studies have recently been generalized in [Daymude et al. 2021], and it is currently one of the most promising models for PM. It relies on explicit communication (message passing) between particles, allowing them to move in coordinated ways, form complex structures, and perform various tasks. The most notable feature of the Amoebot model is the solvability of the leader election; the communication between particles is a very useful capability to solve the leader election problem.

In many cases, the Amoebot model uses the leader elected among the particles to solve the pattern formation problem [Luna et al. 2020, Derakhshandeh et al. 2015]. The ability to elect the leader under the various assumptions is one of the strongest computational power of the Amoebot model.

The SILBOT model [D’Angelo et al. 2020] (as in Silent robot) relaxes several assumptions, especially communication capacity. G. D’Angelo et al. show that a leader election, at most three leaders could be elected due to the symmetry of the initial configuration, can be achieved by this model without any communication from any simple (i.e., without holes) connected configuration. As one of the features of SILBOT, it assumes 2-hop visibility; each particle can detect nodes within 2 hops. To our knowledge, the only leader election problem is investigated under the SILBOT model [D’Angelo et al. 2020], however, this study shows that the leader election is solvable without any explicit communication under some specific model. Although it is difficult to compare the computational power between Amoebot and SILBOT, it is natural to say that Amoebot is stronger than SILBOT if there is no additional assumption.

The MOBLOT model is a computational model based on the LCM model; thus, robots are silent (without explicit communication) and do not have special symmetry-breaking capabilities like SILBOT. Therefore, MOBLOT is the model closest to our proposed model Pairbot model: These two computational models are designed with the goal of significantly changing their computational power by adding a few assumptions to conventional mobile robots. MOBLOT is a model inspired by molecules in which oblivious autonomous mobile robots with some different roles are combined to act as a new large computational entity; such as atoms, molecules are formed. The robots in the same molecules can operate together, e.g., they have geometric agreement, and this enables the robots to break their symmetry because the robots forming molecules become nonequivalent (i.e., there is no symmetry). The literature [Cicerone et al. 2021] gives one special case study of pattern formation, named matter formation, to form molecules by robots. However, this study provides us with the fact that this model can enlarge the class of achievable patterns of oblivious mobile robots that break the symmetry of the system. Since MOBLOT is based on conventional mobile robots, it is a relatively weak computational model compared to the two models above, but assumes relatively strong capabilities (even though they are not unrealistic) compared to our proposed model; Pairbot model assumes only the unique pair of each robot, which can be easily implemented (or simulated) with some previous assumptions, such as lights.

The Uniqueness of Pairbot model: What separates Pairbot model apart is its minimalistic approach to computational capabilities, namely the unique pairing of each robot. The real strength of Pairbot model lies in its connectivity, even under asynchronous schedulers, as shown in our solutions to the problems of perpetual marching and 7-pairbots-gathering. The former illustrates that a problem unsolvable by LCM-robots under a semi-synchronous scheduler is, in fact, solvable by pairbots under an asynchronous scheduler. The latter shows how Pairbot model outperforms traditional robots in terms of visibility range. This clearly highlights the superior efficiency of Pairbot model in solving problems with fewer assumptions, offering advantages in terms of implementability and real-world applicability.

In summary, while Amoebot offers the most computational power, followed by SILBOT and MOBLOT (these two models are hard to compare), Pairbot model carves out its own unique space by efficiently solving problems with fewer computational assumptions. This makes it an ideal choice for scenarios where computational simplicity and efficiency are critical.
1.4 Contribution

In this paper, we introduce a novel method for implementing programmable matter, designed to simplify the exploration and analysis of its capabilities using established knowledge. Our model, known as Pairbot model, features two robots that operate collaboratively as a pair on a triangular grid. The Pairbot model introduces a unique feature called an exclusive move, which is absent from the traditional LCM model. Despite this addition, Pairbot model largely retains the functionalities found in the LCM model. This newly added capability significantly enhances computational power, specifically problem solvability, and offers valuable insights into the realization of programmable matter based on the LCM model.

In Pairbot model, each robot is uniquely paired with another robot, referred to as buddy. We denote these paired robots as pairbot, and a pairbot system comprises two or more pairbots. Every robot in a pairbot can identify its buddy. The paired robots repeatedly alter their geometric relationships, short and long, to achieve their objectives. In the short state, both robots share the same spatial point, while in the long state, they occupy adjacent points.

To elucidate the functioning of pairbot, we present two challenges: the perpetual marching and the 7-pairbots-gathering problems. The former problem is not solvable by LCM-robots under a semi-synchronous scheduler, and the latter is not solvable by LCM-robots with visibility range 1. We propose two deterministic algorithms as solutions for these challenges, which serve to demonstrate how pairbots operate and both the computational power and the limitations of the Pairbot model. In particular, the first algorithm solves the perpetual marching problem under an asynchronous scheduler, and the second solves the 7-pairbots-gathering problem by pairbots with visibility range 1.

1.5 Paper Organization

The rest of this paper is organized as follows: Section 2 introduces the proposed system model, called Pairbot model; Section 3 gives the perpetual marching problem and proposes an algorithm to solve this problem; Section 4 discusses another problem, called the 7-pairbots-gathering problem and an algorithm to solve this problem; and finally Section 5 concludes the paper.

2 Proposed Model: Pairbot model

2.1 Triangular Grid Plane

We consider a set of \( n \) autonomous mobile robots denoted by \( R = \{r_1, r_2, \ldots, r_n\} \) on a two-dimensional triangular grid plane \( \mathbb{T} = \mathbb{Z}^2 \) (Figure 1). The distance between two points \( u \) and \( v \) in \( \mathbb{T} \) is defined by the following equation:

\[
\text{dist}(u, v) = \begin{cases} 
|u.x - v.x| + |u.y - v.y| & \text{if } (u_x - v_x)(u_y - v_y) \geq 0 \\
\max(|u.x - v.x|, |u.y - v.y|) & \text{otherwise}
\end{cases}
\]

If the distance between two points is one, the two points are adjacent. The triangular grid plane \( \mathbb{T} \) can also be represented as an infinite regular graph \( G_\mathbb{T} = (V_\mathbb{T}, E_\mathbb{T}) \), where \( V_\mathbb{T} \) consists of all points on \( \mathbb{T} \) and \( E_\mathbb{T} \) is defined by all two adjacent points on \( \mathbb{T} \) (i.e., for \( u, v \in V_\mathbb{T}, (u, v) \in E_\mathbb{T} \) if \( \text{dist}(u, v) = 1 \).
2.2 Geometric Agreement

Every robot has its own local coordinate system that can be defined by directions (i.e., $X$ and $Y$ axes) and orientations (i.e., positive and negative sides) of each axis. We can consider some levels of consistency among robots on their local compass: total agreement, when all robots agree on the directions and orientations of both axes; partial agreement, when all robots agree on the direction and orientation of only one axis or on the chirality, which means a sense of axis orientation (i.e., clockwise or counter-clockwise); or no agreement, when no agreement exists among the local coordinate system of each robot. In the problems considered in this paper, we assumed no agreement and total agreement respectively. Note that the latter means that all robots in $R$ agree on the directions and orientations of both axes, but no robot agrees on the position of the origin. In other words, no robot knows its global coordinate, but all agree on the sense of direction (e.g., north, south, east, and west).

2.3 Pairbot

In Pairbot model, every robot has its unique partner, called buddy: robot $r_i$ is the buddy of robot $r_j$ if and only if robot $r_j$ is the buddy of robot $r_i$. In this case, we call the two robots $r_i$ and $r_j$ a pairbot. The buddy of each robot is initially determined and never changed. Obviously, the number of robots $n$ is an even number.

2.3.1 Operations

Each robot $r_i$ cyclically performs the following three operations: Look, Compute, and Move based on a well-known computational model (LCM model [Suzuki and Yamashita, 1999]).

- **Look**: Each robot takes a snapshot consisting of robots within the visibility range with respect to its local coordinate system.
- **Compute**: Each robot performs a local computation based on the snapshot taken by the Look phase according to Algorithm $\mathcal{A}$. As a result of the Compute phase, each robot determines its destination point to move.
- **Move**: Based on the result of the Compute phase, each robot actually moves to the adjacent destination point from the current point in the Move phase. A null movement (staying) is allowed (i.e., a robot does not move).

2.3.2 Local Label

Each robot locally maintains the labels for each incident edge (from $\ell_1$ to $\ell_6$) to distinguish its adjacent points. Each robot selects one adjacent point and labels the point as $\ell_0$, and labels the other points from $\ell_2$ to $\ell_6$ in clockwise order (based on its local chirality). The label $\ell_0$ represents its current point. Note that the direction located the point labeled by $\ell_1$ may be different by the assumption of geometric agreements among robots. Moreover, the clockwise order may also vary by the assumption of chirality.

Figure 2 represents an example of the local labels of three robots, $r_i$, $r_j$, and $r_k$, where there is no agreement among the robots. In this figure, robot $r_i$ labels the point on the right side as $\ell_1$; however, robot $r_j$ or $r_k$ labels the point at a different direction with $r_i$’s as $\ell_1$ because robots do not agree on the directions and orientation of any axis. Moreover, the edges of robot $r_k$ are labeled in counterclockwise order, because neither do the robots agree on chirality.

In the problems discussed later in this paper, we assume either no agreement or total agreement depending on the problems.
2.3.3 Capabilities

All robots are oblivious (i.e., do not have memory), which means that they do not know any of their past executions.

Each robot can recognize other robots within visibility range $V$. This means that each robot can sense only up to $V$ hops far from itself. In other words, the robot $r_u$ at the point $u$ and the robot $r_v$ on point $v$, where $r_u, r_v \in R$ and $u, v \in V_T$, can observe each other only when $\text{dist}(u, v) \leq V$. We assume here that the visibility range of each robot is one (i.e., each robot can observe only the robots at the adjacent points).

We also assume that each robot has the capability of weak multiplicity detection, that is, each robot can distinguish from the following three cases at points within its visibility: no robot exists; only one robot exists; and more than one robot exists. Note that if each robot can count the exact number of robots that occupy the point within its visibility range, this is strong multiplicity detection.

All robots in $R$ are anonymous: each robot has no identifier and no robot is distinguishable by its appearance. However, each robot maintains the position of its buddy using its local label, implying that each has a local memory to maintain the position (local label) where its buddy is located.

We further assume here that when a pairbot occupies the same point, only one robot can move to its adjacent point when it moves. We call this an exclusive move. Note that no robot knows if any other two robots are pairbot or not.

Figure 3: Two states (positional relation) of two robots in the same pair

The two robots in one pairbot occupy the same point or two adjacent points on $T$ (Figure 3). We call the former one a short state and the latter a long state. When a pairbot is in a short state, only one robot in the pairbot can (exclusively) move the adjacent point from its current point, and the pairbot becomes a long state. When a pairbot is in a long state, either of the two robots can move to the point occupied by its buddy, and the state is changed back to a short state. In the Pairbot model, every pairbot repeatedly changes its state to short and long to achieve the goal. A pairbot never knows which robot moved while its state changes because all robots are oblivious.

Figure 4 illustrates an example of pairbots. Pairbot $r_i$ and $r_j$ is in a long state, and some other robots are placed at the other points in $T$. In this case, robot $r_i$ recognizes that its buddy is robot $r_j$ on its right side. However, $r_i$ cannot know the pair relations of the other robots (e.g., robot $r_i$ never knows the buddy of robot $r_x$ on its lower left side). The robot $r_y$ can recognize that another robot is at the same point due to weak multiplicity detection. Furthermore, $r_y$ can know that the buddy $r_x$ is on the point of its upper left side and is in a long state.
2.4 Scheduler

We consider a scheduler that decides which robot to activate and the timing of each operation. A scheduler has three representative assumptions (models): fully-synchronous (FSYNC); semi-synchronous (SSYNC); and asynchronous (ASYNC).

In an FSYNC scheduler, all robots are activated at the same time, and the three operations of Look, Compute, and Move are executed based on exactly the same cycle time. In an SSYNC scheduler, all robots perform their operations at the same time, but some may not be activated. The robots, which are not activated by a scheduler, wait until all activated robots terminate their operations. Lastly, in an ASYNC scheduler, no assumption on the cycle time of each robot is provided, implying that all robots execute their operations at unpredictable time instants and durations.

We assume herein an ASYNC scheduler such that the Move phase operates atomically: each robot requires an unpredictable finite time to operate in the Look or Compute phase, but it can atomically move in a property called the move-atomic property. Therefore, each robot is never observed while moving. We assume that the two robots, which are a pairbot are activated at the same time by the scheduler.

2.5 Configuration

A configuration \( C_t \) consists of the positions of all robots in \( R \) at time \( t \):
\[
C_t = \{(r_1.x(t), r_1.y(t)), (r_2.x(t), r_2.y(t)), \ldots, (r_n.x(t), r_n.y(t))\}
\]
where \((r_i.x(t), r_i.y(t))\) is the global coordinate of robot \( r_i \) at time \( t \) on \( T \). Note that no robot knows its global coordinate on \( T \). Let \( R' \) be a non-empty subset of \( R \) and let \( \mathcal{A} \) be an algorithm. We denote \( C_t \rightarrow (R', \mathcal{A}) C_{t+1} \) if a configuration \( C_{t+1} \) is obtained when each robot in \( R' \) simultaneously performs its Move operation of \( \mathcal{A} \) in configuration \( C_t \). Hence, a scheduler can be presented as an infinite sequence \( R_1, R_2, \ldots \) of nonempty subsets of \( R \). An execution \( \Xi_{\mathcal{A}}(S, C_0) \) of Algorithm \( \mathcal{A} \) along the schedule \( S = R_1, R_2, \ldots \) starting from configuration \( C_0 \), where \( C_0 \) is an initial configuration, can be defined as the infinite sequence of configurations \( C_0, C_1, \ldots \) such that \( C_i \rightarrow (R_{i+1}, \mathcal{A}) C_{i+1} \) for all \( i \geq 0 \).

3 The Perpetual Marching Problem

3.1 Problem Definition

In this section, we consider the perpetual marching problem, which is an infinite linear movement of pairbots without any geometrical agreement (i.e., they do not have any common sense of direction).

We now define the marching problem as follows:

**Definition 1 Perpetual Marching Problem.** Algorithm \( \mathcal{A} \) solves the perpetual marching problem if there is a well-initiated configuration \( C_{in\text{it}} \), a certain direction \( D \), a constant \( k \), and an infinite execution \( \Xi_{\mathcal{A}}(S, C_{in\text{it}}) \) such that after each (constant) \( t \) steps, all robots move to a point that is a distance of \( k \) toward the direction \( D \).

Intuitively, when locating the robots in a predetermined configuration (a well-initiated configuration \( C_{in\text{it}} \)), an algorithm makes all the robots move infinitely (an infinite execution) at a constant speed (a constant \( k \)) in a specific direction (a certain direction \( D \)). Figure 5 illustrates an example of perpetual marching.

It is worthwhile to noting that the perpetual marching requires a fixed distance movement within a constant speed. In other words, a movement in a specific direction within a finite time is not feasible perpetual marching. For instance, a movement such as a perpetual exploration while expanding the range (refer to Figure 6(a)) or a repetitive movement back and forth (refer to Figure 6(a)) is not feasible in the perpetual marching problem, because from the perspective of moving in a fixed direction, their speed progressively decreases. On the other hand, whether the movement is in a straight line (refer to Figure 7(a)) or follows a zigzag path, (refer to Figure 7(b)) it becomes a feasible perpetual marching as long as it maintains a constant speed.

3.2 Perpetual Marching Algorithm

First we introduce an well-initiated configuration \( C_{in\text{it}} \) as Figure 8. Figure 9 shows the proposed algorithm (using graphical representation) consisting of 7 rules when configuration \( C_{in\text{it}} \) is given. The detailed pseudocode of the algorithm is provided in the next section. In each rule, the circle in the center represents the grid point where a robot (who executes the algorithm) exists and 6 other circles present adjacent grid points. Each number shows the number of robots and a black circle represents the point where the buddy exists. It is worth to noting that even if there are three or more robots at one point, the number (of robots) becomes 2 due to the weak multiplicity. Moreover, these rules are
applied as the same rules even mirrored and/or rotated because robots do not agree on any axis nor chirality. If the observation result of the robot satisfies any of these conditions, the robot moves to the point indicated by a black arrow. Note that the direction to move is uniquely determined in each rule, thus every robot can move to the correct direction without a sense of direction.

During the execution of the algorithm, eight different configurations appear sequentially. After 8 steps, the initial configuration appears again with a shift of distance one to the right direction. These 8 steps are infinitely repeating and the perpetual marching is achieved. As a result, the following corollary holds.

**Lemma 1** The proposed algorithm solves the perpetual marching problem under an ASYNC scheduler.

**Proof 1** In configuration $C_{\text{init}}$, there exists only one pairbot satisfying a rule (rule 1). Therefore, even under an ASYNC scheduler, only one pairbot can move. After one step, another configuration appears; however, in this configuration, only one pairbot satisfies a rule (rule 2). In all subsequent configurations, only one pairbot can move. After 8 steps, a configuration that is a translation of the configuration $C_{\text{init}}$ appears, which means that all robots move to the adjacent point at the same direction. Figure 10 shows the robot that satisfies a rule (a white robot), and the rule number can be applied in each configuration.
Figure 7: Examples of feasible perpetual marching

Figure 8: An well-initiated Configuration $C_{\text{init}}$

Figure 9: The proposed algorithm for perpetual marching
As a result, in every configuration, the proposed algorithm moves only one robot, and eight different configurations sequentially appear while moving one hop in a certain direction. This implies that the proposed algorithm solves the perpetual marching problem.

By the proof of Lemma 1, the following corollary holds.

**Corollary 1** The proposed algorithm moves all the robots to a predetermined direction (by an initial configuration) with a distance of 1 in 8 steps.

Here the term steps means the total number of robots that moves. By Lemma 1 and Corollary 1, the following theorem holds.

**Theorem 1** When a well-initiated configuration is given, the proposed algorithm solves the perpetual marching problem by moving all the robots to the same direction with a distance of 1 in every 8 steps under an ASYNC scheduler.

Note that the proposed algorithm is optimal in terms of the number of pairbots; impossibility results for 1 pairbot or 2 pairbots can be easily proven because the total number of (feasible) configurations is small enough.

### 3.3 Pseudocode of the Perpetual Marching Algorithm

In this section, we present the pseudocode of the proposed algorithm to solve the perpetual marching by pairbots without a sense of direction.

Algorithm 1 shows the proposed algorithm. The proposed algorithm consists of 7 rules; each rule is presented as [Rule No.]: [Condition] → [Action], which means that a robot executes an [Action] when [Condition] in the same [Rule No.] is satisfied. Since robots do not agree on any axis nor chirality, all conditions in the algorithm are rotatable and reflectable; which means that the condition for each rule becomes TRUE if the observed result of the robot satisfies the condition when rotated and/or mirrored in any direction.

### 3.4 Impossibility Result of Perpetual Marching by LCM-Robots

Now we discuss the perpetual marching problem also in LCM model, with no geometric agreement (i.e., without a sense of directions). If we assume an FSYNC scheduler, the problem can be easily solved by 3 autonomous mobile robots with visibility range 1 using the following algorithm.

- Initial configuration $C_{init}$: Let $u$ and $v$ be the two adjacent points. The robot $r_1$ is located $u$ and the robots $r_2$ and $r_3$ are located $v$.  

![Figure 10: Execution of the proposed algorithm](image)
Algorithm 1 Algorithm for perpetual marching

variables and functions:
- Buddy \( \in \{\ell_0, \ell_1, \ldots, \ell_6\} \): A label where its buddy exists.
- Chk(\(\ell_x, Nr\)) : A function for checking the number of robots that returns TRUE when the number of robots on the adjacent point at \(\ell_x\) is the same as Nr. Note that Nr \(\in \{0, 1, 2\}\) due to weak multiplicity detection. The first parameter can be a set of labels \(L \subseteq \{\ell_x|0 \leq x \leq 6\}\); the function returns TRUE only when Chk(\(\ell_x \in L, Nr\)) = TRUE.
- Move(\(\ell_x\)) : Move to \(l_x\).

algorithm:
/* All conditions are rotatable and reflectable:
the condition for each rule becomes TRUE if the observed result of the robot satisfies
the condition when rotated and/or mirrored in any direction.*/
Rule1: Buddy = \(\ell_0 \land \text{Chk}(\{\ell_2, \ell_3, \ell_4, \ell_5, \ell_6\}, 0) \land \text{Chk}(\ell_1, 1) \rightarrow \text{Move}(\ell_1)\)
Rule2: Buddy = \(\ell_0 \land \text{Chk}(\{\ell_1, \ell_2, \ell_3, \ell_4\}, 0) \land \text{Chk}(\ell_2, 1) \land \text{Chk}(\ell_1, 1) \rightarrow \text{Move}(\ell_1)\)
Rule3: Buddy = \(\ell_1 \land \text{Chk}(\{\ell_2, \ell_3, \ell_4\}, 0) \land \text{Chk}(\{\ell_0, \ell_1, \ell_6\}, 1) \land \text{Chk}(\ell_5, 2) \rightarrow \text{Move}(\ell_1)\)
Rule4: Buddy = \(\ell_0 \land \text{Chk}(\{\ell_2, \ell_3, \ell_4, \ell_5, \ell_6\}, 0) \land \text{Chk}(\ell_3, 2) \land \text{Chk}(\ell_4, 1) \rightarrow \text{Move}(\ell_1)\)
Rule5: Buddy = \(\ell_1 \land \text{Chk}(\{\ell_2, \ell_3, \ell_4, \ell_5, \ell_6\}, 0) \land \text{Chk}(\{\ell_0, \ell_1\}, 1) \rightarrow \text{Move}(\ell_1)\)
Rule6: Buddy = \(\ell_0 \land \text{Chk}(\{\ell_1, \ell_2, \ell_3, \ell_4\}, 0) \land \text{Chk}(\ell_2, 1) \land \text{Chk}(\ell_4, 2) \rightarrow \text{Move}(\ell_1)\)
Rule7: Buddy = \(\ell_0 \land \text{Chk}(\{\ell_3, \ell_5, \ell_6\}, 0) \land \text{Chk}(\ell_2, 1) \land \text{Chk}(\ell_4, 2) \rightarrow \text{Move}(\ell_2)\)

- Algorithm: If a single robot observes two accompanied robots, it moves to the point occupied by them. If an accompanied robot observes a single robot, it moves to the point opposite to it.

Two pivotal questions naturally emerge:
1. "Is the perpetual marching problem solvable by LCM-robots under either an SSYNC or an ASYNC scheduler?"
2. "What is the minimum number of LCM-robots required to solve the perpetual marching problem in the LCM model under either an SSYNC or an ASYNC scheduler?"

To partially address these queries, we show the following theorem in this section.

Theorem 2 Under an SSYNC scheduler, no deterministic algorithm exists that can solve the perpetual marching problem using unoriented (i.e., lacking geometric agreement) LCM-robots with a visibility range of 1, provided the number of robots is six or fewer.

Now we prove Theorem 2 in the following subsections.

3.4.1 Preliminaries

Let \(R = \{r_1, r_2, \ldots, r_6\}\) be a set of 6 autonomous mobile robots with visibility range 1. The robots do not have any geometrical agreement (i.e., they do not agree on any axis nor chirality). Moreover, we assume that each robot has the capability of weak multiplicity detection. Here we consider an SSYNC scheduler.

A configuration \(C_i\) consists of all positions of all robots in \(R\), and we call two configurations \(C_i\) and \(C_j\) different when \(C_i\) cannot be transformed into \(C_j\) through translation, mirroring, and/or rotation. We consider a set of all connected configurations consisting of 6 autonomous mobile robots \(C = \{C_1, C_2, C_3, \ldots, C_n\}\) such that configurations \(c_i\) and \(c_j\) for any \(i \neq j\) are different configurations. A configuration \(C_i\) is a \(k\)-point configuration if exactly \(k\) points are occupied in \(C_i\). Let \(C^k\) be a set of all \(k\)-point configuration in \(C\); \(C^1 \cup C^2 \cup C^3 \cup C^4 \cup C^5 \cup C^6 = C\) and \(C^1 \cap C^2 \cap C^3 \cap C^4 \cap C^5 \cap C^6 = 0\). Figure 11 shows the examples of \(k\)-point configurations by 6 robots.

Algorithm \(\mathcal{A}\) consists of a set of rules; a rule Rule\(i\) is defined as a tuple (View\(_i\), Dest\(_i\)) which means that a robot moves to Dest\(_i\), if its observation result is the same as View\(_i\). Note that the number of the possible observation results is finite (because the visibility range is assumed to 1), the number of the possible actions is also finite (because each robot can move to a point among its 6 adjacent points). Let \(\mathcal{A} = \{\text{Rule}1, \text{Rule}2, \ldots, \text{Rule}m\}\) be a set of all possible rules. Algorithm \(\mathcal{A}\) can be defined as a set of (non-conflicted) rules \(\mathcal{A}_\mathcal{A} \subset \mathcal{A}\).

Now we consider an algorithm-based directed graph \(\overrightarrow{G}_\mathcal{A} = (V, A_\mathcal{A})\) where \(V = C \cup \{C_{\text{distConf}}\}\). Arc (i.e., directed edge) \((C_i, C_j) \in E_\mathcal{A}\) if \(C_j\) is obtained when a robot performs its Move operation according to a rule in algorithm
Figure 11: Examples of $k$-point configurations
We call a rule such that when a robot executes the rule, all the robots located at seven points within the visibility range of the robot become disconnected a \textit{SSYNC} rule. Moreover, arc (i.e., directed edge) $(C_i, C_{\text{disCon}}) \in E_{A\mathcal{F}}$ if a non-connected configuration is obtained when a robot executes a rule in $\mathcal{A}$ in configuration $C_i$.

3.4.2 Basic Strategy of the Proof

First, we present the basic strategy of the proof. We assume an SSYNC scheduler, this means that in each round, the scheduler determines a set of robots and they executes the (same) algorithm. If there exists an algorithm to solve the perpetual marching problem, it must be able to handle all patterns from all possible schedulers. Additionally, since the robots do not have a sense of direction, the destination point a robot moves to may not be uniquely determined depending on the rule in the algorithm. An algorithm for the perpetual marching problem must work correctly even when the destination is arbitrarily selected. Therefore, in the proof of impossibility, we assume that both the scheduler and the destination are chosen adversarially, and show that there is no algorithm can solve the perpetual marching problem.

In the proof of impossibility, we assume the following adversarial scenarios:

1. When there are two robots satisfying View, and if they simultaneously execution of Rule makes the perpetual marching impossible, the scheduler selects these two robots and makes them act in a way that makes the perpetual marching impossible.
2. Except for 1, the scheduler always selects only one point where there exists a robot satisfies any rule in the algorithm, and make all the robots at the point simultaneously execute the algorithm (i.e., the scheduler never selects two different robots at two different points).
3. In the case of 2, when the destination is arbitrarily selected, all robots choose the same direction. In other words, if multiple robots are on the same point, they always move as a group and never scatter during execution.
4. If a destination is not uniquely determined by a rule, the scheduler selects destinations in a way that results in unfavorable configuration transitions (e.g., if a specific movement of some robots causes a disconnected configuration, the scheduler selects that destination).

We generate digraph $G_{A\mathcal{F}}$ as a configuration transition diagram under the above scenario to prove the impossibility. If there is algorithm $\mathcal{A}$ to solve the perpetual marching problem, digraph $G_{A\mathcal{F}}$ includes a directed cycle (denote $C_f = (C_i, C_j, C_k, \ldots , C_i)$) presenting an infinite execution (i.e., infinite transition of configurations) of algorithm $\mathcal{A}$.

Remind that every robot’s visibility range is 1 and the robots do not agree on any axis nor chirality, thus a connected configuration cannot be obtained from any disconnected configuration due to an adversarial scheduler. Therefore, the following corollary holds.

\textbf{Corollary 2} If there exists directed cycle $\tilde{C}_f$ in digraph $G_{A\mathcal{F}}$, $\tilde{C}_f$ consists of only connected configurations.

We call an infinite execution of algorithm $\mathcal{A}$ from an well-initiated configuration (i.e., any configuration appeared in $\tilde{C}_f$) the execution of $\tilde{C}_f$.

Moreover, we call directed cycle $\tilde{C}_f$ in digraph $G_{A\mathcal{F}}$ is illegible, if $\tilde{C}_f$ satisfies any of the following conditions.

1. \textbf{(No movement)} Let $C_i$ be a well-initiated configuration in $\tilde{C}_f$. After the constant number of executions, when $C_i$ appears again (denote as $C_i^2$), all the robots are at the exactly same points in $C_i$ and $C_i^2$.
2. \textbf{(Faulty concurrency)} In configuration $C_i$ in $\tilde{C}_f$, there are two different robots ($r_x$ and $r_y$) and two different rules (Rule$_{r_x}$ and Rule$_{r_y}$ in $\mathcal{A}$) such that robot $r_x$ (resp. $r_y$) can execute Rule$_{r_x}$ (resp. Rule$_{r_y}$). When the two robots simultaneously execute $\mathcal{A}$, configuration $C_j$ which is not included in $\tilde{C}_f$ is obtained.
3. \textbf{(Adversarial transition)} Let $C_i$ be a configuration in $\tilde{C}_f$, and $C_j$ be a configuration NOT in $\tilde{C}_f$. There exists arc $(C_i, C_j)$ in $A_{A\mathcal{F}}$.

3.4.3 Proof of Impossibility Result

Let $\mathcal{A}$ be an algorithm to solve the perpetual marching problem by autonomous mobile robots with visibility range 1 under an SSYNC scheduler.

We call a rule such that when a robot executes the rule, all the robots located at seven points within the visibility range of the robot become disconnected a \textit{locally disconnecting rule}.
Figure 12 shows some examples of locally disconnecting rules. In each figure, a hexagon represents a local view of the observing robot (white circle) and each arrow shows a destination to move when the robot obtains such an observation result. Depending on the configuration, even if some rule (e.g., Figure 12(c)) is executed, there are cases where the next configuration remains connected. However, in some configurations, it is possible for it to become disconnected. Since the robots executing the rules cannot perceive configurations outside their visibility range, adding locally disconnecting rules to the algorithm $A$ may result in execution even in cases where it becomes disconnected, leading to disconnected configurations. From the above fact and Corollary 2, the following lemma holds.

**Lemma 2** There is no locally disconnecting rule in algorithm $A$.

Now we have prepared the groundwork to proceed with the proof of impossibility. We show the impossibility by showing that none of the vertices in the graph are included in $\vec{C}_f$. Specifically, we start by proving that none of the points in all 1-point configurations ($\in C_1$) are included, and then consider the points in all 2-point configurations ($C_2$). In any 2-point configuration, we utilize the fact that the algorithm does not contain rules that transform the configuration into any 1-point configuration to reduce the number of vertices to consider. This is achieved by ensuring that when two robots at different points move to the same point once, the scheduler always activates these two robots simultaneously, providing the same view and ensuring that they always perform the same operations (remind that we consider a deterministic algorithm). This implies that if configuration $C_i$ in $C^k$ is obtained from configuration $C_j$ ($i \neq j$) in $C^{(k+1)}$ by an algorithm, configuration $C_i$ cannot be obtained again by the algorithm due to an adversarial scheduler. Hence the following corollary holds.

**Corollary 3** If there exists directed cycle $\vec{C}_f$ (i.e., algorithm to solve the perpetual marching exists), all the configurations included in $\vec{C}_f$ are the same $k$-point configurations (i.e., $\forall C_i, C_j \in \vec{C}_f: \{C_i, C_j\} \subseteq C^k$).

From the definition of $k$-point configuration, there is only one configuration included in $C^1$. Figure 13 presents the unique configuration of 1-point configuration ($C_1$).

**Lemma 3** Let $C_1$ be the unique configuration in $C^1$. Cycle $\vec{C}_f$ never includes $C_1$.

**Proof 2** Configuration $C_1$ is the same as the configuration depicted in Figure 11(a). This implies that all the robots have the same observation result. Figure 14 represents the observation result of every robot in $C_1$ (the notation of the rule is the same as the graphical representation of the algorithm for perpetual marching; refer to Section 3.2). The destination of the rule presented in Figure 14 is the point on the right, however, the destination point is irrelevant in this
rule. In other words, this means that the rule remains the same regardless of where the destination point is because the robots do not agree any axis. We call this rule Rule 1.

Assume that configuration $C_1$ is included in $C_f$. If algorithm $A$ includes Rule 1, the scheduler activates all the 6 robots simultaneously, and makes them to move to the same point. After that, the scheduler activates all the robots again, and makes them to return back to the previously located point. By repeating this, no robot cannot move to any other point than these two points. As a result, the perpetual marching cannot be solved. If algorithm $A$ does not include Rule 1, no robot cannot move. Therefore, there is no algorithm to solve the perpetual marching problem from configuration $C_1$, and this implies that configuration $C_1$ is not included in $C_f$.

**Lemma 4** Cycle $C_f$ never includes any configuration in $C_2$.

![Figure 15: Three configurations in $C_2$](image)

**Proof 3** Figure 15 shows all the 2-point configurations ($C_2$). Note that all the robots in any of two configuration $C_2$ and $C_3$ have the same observation results due to the weak multiplicity.

![Figure 16: Three views of the robots in $C_2$](image)

Figure 16 presents the view can be observed by the robots in 2-point configuration (we call these three views View$_2$, View$_3$, and View$_4$ respectively). In any configuration in $C_2$, each robot obtains one of the observation result among View$_2$, View$_3$ and View$_4$. By Lemma 2, considering any locally disconnecting rule is not required, thus only the three rules depicted in Figure 17 have to be considered.

![Figure 17: Three feasible rules in $C_2$](image)
Assume that configuration $C_2^1$ is included in $\vec{C}_f$. If algorithm $A$ includes Rule2 (resp. Rule3), the scheduler activates a single robot (resp. all 5 robots located at the same point). And the same configuration ($C_2^1$) appears again (refer to Figure 18). In the next activation, the scheduler can make the single robot (or the 5 robots) to move back to the previously occupying point. Thus configuration $C_2^1$ is not included in $\vec{C}_f$.

Now assume that configuration $C_2^2$ and/or $C_3^2$ is included in $\vec{C}_f$. In this case, only View4 can be obtained by the robots. If algorithm $A$ includes Rule4, the scheduler can activate all the 6 robots, and make them to move to the same point causing 1-point configuration by faulty concurrency. Therefore, configurations $C_2^2$ and $C_3^2$ are not included in $\vec{C}_f$.

Lemma 5 Cycle $\vec{C}_f$ never includes any configuration in $C^3$.

Proof 4 First, we present all the configurations included in $C^3$ as Figure 19. There are 15 3-point configurations, we denote the configurations by $C_3^1$ to $C_3^{15}$; $C_3^3 = \{C_3^1, C_3^2, \ldots, C_3^{15}\}$.

From configurations $C_3^1$ to $C_3^{15}$, we can enumerate all the views can be obtained by any robots in $C^3$. Figure 16 shows all the views in any configuration in $C^3$ (19 views exist). Note that views View2, View3, and View4 are the same as them appeared in $C^2$.

From all the views obtained by a robot in any configuration in $C^3$, we can enumerate all feasible rules. Also in this case, we do not consider a locally disconnecting rule (due to Lemma 2). Moreover, we also exclude a rule that makes a robot to any point occupied by another robot because such a rule causes a configuration in $C_1^3$ or $C_2^3$ (refer to Corollary 3).

As a result, we obtain the 16 rules depicted in Figure 21.

Now we can obtain the induced subgraph $G_{A_{II}}(C^3)$ of $G_{A_{II}}(C^3)$ because the set of arcs in $G_{A_{II}}(C^3)$ is defined by the rules in Figure 21. It is worthwhile to mention that no robot can distinguish the difference between $C_3^6$ and $C_3^7$, and between $C_3^9$ and $C_3^{10}$ due to the weak multiplicity.

Figure 22 presents the induced subgraph $G_{A_{II}}(C^3)$. A dotted arc represents an execution occurring a disconnected configuration or a smaller k-point configuration due to a faulty concurrency. There are 6 (directed) loops at configurations $C_3^3$, $C_3^4$, $C_3^5$, $C_3^{10}$, $C_3^{11}$, and $C_3^{14}$. An infinity execution of these loops are the same; only the robots located at the center point repeatedly move back and forth between two points. Thus these loops cannot become cycle $\vec{C}_f$.

Now we consider the following two (directed) cycles, $(C_3^1 \rightarrow C_3^3 \rightarrow C_3^1)$ and $(C_3^6 \rightarrow C_3^{10} \rightarrow C_3^9)$, and $(C_3^8 \rightarrow C_3^9 \rightarrow C_3^8)$. However in any cycles, only some robots at the same point infinitely move back and forth between two adjacent points. This implies that there exists a robot which never moves, hence, these cycles cannot become cycle $\vec{C}_f$. 

16
Figure 19: All the configurations in $C^3$
We can prove the remaining parts of the proof below using the same manner, but each proof becomes much more longer, thus we omit the detailed proofs of the following three lemmas for $C_4$, $C_5$, and $C_6$ respectively here.

**Lemma 6** Cycle $\vec{C}_f$ never includes any configuration in $C_4$.

**Proof 5** (Sketch) There are 47 configurations in $C_4$, and 40 views can be obtained by a robot in any configuration in $C_4$. From these views, we can obtain 36 feasible rules in $C_4$. A resultant induced subgraph $G_{\rightarrow A_1}(C_4)$ of $G_{A_1}$ is given in Figures 23 and 24. There are many cycles in $G_{\rightarrow A_1}(C_4)$, however, no appropriate cycle in $G_{A_1}(C_4)$ which can become feasible $\vec{C}_f$. 

Figure 20: All views can be obtained by a robot in $C_3$.
Lemma 7  Cycle $\mathcal{C}_f$ never includes any configuration in $C^5$.

Lemma 8  Cycle $\mathcal{C}_f$ never includes any configuration in $C^6$.

We omit the proof of Lemmas 7 and 8 however these are can be proven by the same manner; there exist 87 and 82 configurations in $G_{A1}(C^5)$ and $G_{A1}(C^6)$ respectively.

From lemmas 3, 4, 5, 6, 7, and 8 the following lemma holds.

Lemma 9  There is no directed cycle $\mathcal{C}_f$ in $G_{a sf}$ implying the valid infinite execution of the perpetual marching.

Therefore, the following theorem holds by Lemma 9.

Theorem 3  Under an SSYNC scheduler, no deterministic algorithm exists that can solve the perpetual marching problem using unoriented (i.e., lacking geometric agreement) LCM-robots with a visibility range of 1, provided the number of robots is six.

Theorem 3 presents an impossibility result only for 6 LCM-robots, however, an impossibility result for 5 or less LCM-robots can be easily proven. In particular, the impossibility results for 1, 2, or 3 LCM-robots can be proven by a straight-forward manner (only the small number of all connected configurations exist), and the results for 4 or 5

![Figure 21: All feasible rules in $C^3$](image)
Figure 22: Configuration transition in $C^3$ (i.e., the induced subgraph $G_{A^I}(C^3)$ of $G_{A^I}$)
LCM-robots can be proven as the same manner as the case for 6 LCM-robots. However, many of $k$-point configurations appeared when the number of robots is 4 or 5 are also appeared in the proof for 6 LCM-robots, thus, there are only some configurations have to be newly considered to prove the impossibility result. As a result, Theorem 2 holds.
Figure 24: Configuration transition in $C^4$ (i.e., the induced subgraph $G_{Al}(C^4)$ of $G_{Al}$): From $C_{19}^4$ to $C_{17}^4$
4 The 7-pairbots-gathering Problem

4.1 Problem Definition

In this section, we consider the 7-pairbots-gathering problem, which is the gathering problem of 7 pairbots on a triangular grid. Generally, many gathering problems aim to gather all robots to one common point, but here we consider the gathering problem where two or more robots do not exist at the same point like the gathering considered in fat robot models Czyzowicz et al. [2009]. This means that our goal is to gather the robots as close together as possible.

We define the 7-pairbots-gathering problem as the following.

**Definition 2 7-pairbots-gathering Problem.** Given an arbitrary connected configuration consisting of 7 pairbots in short state, algorithm $A$ solves the 7-pairbots-gathering problem if $A$ satisfies all the following conditions: (1) algorithm $A$ eventually terminates; algorithm $A$ eventually reaches a configuration such that no robot can move, (2) all pairbots are in short state and no two pairbots exist at the same point when algorithm $A$ terminates, and (3) $\text{dist}(r_i, r_j) \leq 2$ for any non-paired two robots $r_i$ and $r_j$ when algorithm $A$ terminates.

Figure 25 shows three examples of initial configurations and Figure 26 illustrates the goal configuration of 7-pairbots-gathering problem. Note that only one configuration as Figure 26 is allowed as a goal configuration.

We can also consider the 7-robots-gathering problem which is the gathering by 7 LCM-robots (robots in LCM model) instead of pairbots. This problem is already introduced in Shibata et al. [2022], and the two following theorems hold by the literature.

**Theorem 4 Shibata et al. [2022]** For robots with visibility range 1, there exists no deterministic algorithm to solve the 7-robots-gathering problem even under an FSYNC scheduler.

**Theorem 5 Shibata et al. [2022]** For robots with visibility range 2, there exists a deterministic algorithm to solve the 7-robots-gathering problem from any connected initial configuration under an FSYNC scheduler.

Theorems 4 and 5 suggest that the visibility range plays a critical role in the solvability of the 7-robots-gathering problem, especially under deterministic algorithms and an FSYNC scheduler. While it is unsolvable for robots with a visibility range of 1, the problem becomes solvable for robots with a visibility range of 2 if they start from any connected initial configuration. In the following, we show that the 7-pairbots-gathering problem for pairbots is solvable even with visibility range 1.
4.2 7-pairbots-gathering Algorithm

Here we introduce the proposed algorithm to solve the 7-pairbots-gathering algorithm using graphical representation. The detailed pseudocode of the algorithm will be presented in the next section.

Figures shows the proposed algorithm by pairbots with a common sense of directions. Each white circle shows a pairbot that executes the proposed algorithm, and each black circle represents the observed robots. Each cross mark means a point which is not occupied by any robot, a triangle presents a point occupied one or more robots (in rules 14, 16, 17, and 21), and an inverted (reversed) triangle represents a point occupied two robots or no robot (in rule 41). Some rules for a pairbot in long state marks a asterisk near the pairbot, which means the robot executes the rule even if there are some other robots at the same (i.e., center) point (in rules from 7 to 11, 20, and 21). If there is no any mark at the point, the algorithm does not care about the point.

4.3 Pseudocode of the 7-pairbots-gathering Algorithm

In this section, we present the pseudocode of the proposed algorithm to solve the 7-pairbots-gathering problem. We assume that all robots agree on the orientation and directions of both axes, which is the same assumption as the literature [Shibata et al. 2022]; every robot labels the adjacent point located on its right side as $\ell_1$, and then labels the other adjacent points from $\ell_2$ to $\ell_6$ in clockwise order (refer to the labeling of robot $r_i$ in Figure 2). The point where a robot currently exists is labeled by $\ell_0$. Algorithm 2 shows the proposed algorithm; as the same as the algorithm in the previous section, each rule is presented as [Rule No.]: [Condition] → [Action].

Algorithm 2 Algorithm for 7-pairbots-gathering
variables and functions:
- Buddy $\in \{\ell_0, \ell_1, \ldots, \ell_6\}$: A label where its buddy exists.
- Chk($\ell_x$, Nr): A function for checking the number of robots that returns TRUE when the number of robots on the adjacent point at $\ell_x$ is the same as Nr. Note that Nr $\in \{0, 1, 2\}$ due to weak multiplicity detection. The first parameter can be a set of labels $L \subseteq \{\ell_x|0 \leq x \leq 6\}$; the function returns TRUE only when Chk($\forall \ell_x \in L$, Nr) = TRUE.
- Move($\ell_x$): Move to $\ell_x$.

algorithm:
Rule1: Buddy = $\ell_0 \land$ Chk($\{\ell_2, \ell_3, \ell_4, \ell_5, \ell_6\}$, 0) $\land$ Chk($\ell_1$, 2) $\rightarrow$ Move($\ell_2$)
Rule2: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_3, \ell_4, \ell_5, \ell_6\}$, 0) $\land$ Chk($\ell_2$, 2) $\rightarrow$ Move($\ell_3$)
Rule3: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_4, \ell_5, \ell_6\}$, 0) $\land$ Chk($\ell_3$, 2) $\rightarrow$ Move($\ell_4$)
Rule4: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_3, \ell_5, \ell_6\}$, 0) $\land$ Chk($\ell_4$, 2) $\rightarrow$ Move($\ell_5$)
Rule5: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_3, \ell_4, \ell_6\}$, 0) $\land$ Chk($\ell_5$, 2) $\rightarrow$ Move($\ell_6$)
Rule6: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5\}$, 0) $\land$ Chk($\ell_0$, 2) $\rightarrow$ Move($\ell_1$)
Rule7: Buddy = $\ell_2 \land$ Chk($\{\ell_3, \ell_4, \ell_5, \ell_6\}$, 0) $\land$ Chk($\ell_0$, 1) $\land$ Chk($\ell_1$, 2) $\rightarrow$ Move($\ell_2$)
Rule8: Buddy = $\ell_3 \land$ Chk($\{\ell_4, \ell_5, \ell_6\}$, 0) $\land$ Chk($\ell_0$, 1) $\land$ Chk($\ell_2$, 2) $\rightarrow$ Move($\ell_3$)
Rule9: Buddy = $\ell_4 \land$ Chk($\{\ell_5, \ell_6\}$, 0) $\land$ Chk($\ell_0$, 1) $\land$ Chk($\ell_3$, 2) $\rightarrow$ Move($\ell_4$)
Rule10: Buddy = $\ell_5 \land$ Chk($\{\ell_1, \ell_2, \ell_6\}$, 0) $\land$ Chk($\ell_0$, 1) $\land$ Chk($\ell_4$, 2) $\rightarrow$ Move($\ell_5$)
Rule11: Buddy = $\ell_6 \land$ Chk($\{\ell_1, \ell_2, \ell_3, \ell_4\}$, 0) $\land$ Chk($\ell_0$, 1) $\land$ Chk($\ell_5$, 2) $\rightarrow$ Move($\ell_6$)
Rule12: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_3, \ell_4\}$, 0) $\land$ Chk($\ell_2$, 2) $\rightarrow$ Move($\ell_1$)
Rule13: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_3, \ell_5\}$, 0) $\land$ Chk($\ell_1$, 2) $\rightarrow$ Move($\ell_2$)
Rule14: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_3, \ell_6\}$, 0) $\land$ Chk($\ell_1$, 2) $\rightarrow$ Move($\ell_3$)
Rule15: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_3, \ell_5\}$, 0) $\land$ Chk($\ell_2$, 2) $\rightarrow$ Move($\ell_4$)
Rule16: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_6\}$, 0) $\land$ Chk($\ell_4$, 2) $\rightarrow$ Move($\ell_5$)
Rule17: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_6\}$, 0) $\land$ Chk($\ell_4$, 2) $\rightarrow$ Move($\ell_6$)
Rule18: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_2, \ell_3, \ell_4\}$, 0) $\land$ Chk($\ell_5$, 2) $\rightarrow$ Move($\ell_1$)
Rule19: Buddy = $\ell_2 \land$ Chk($\{\ell_1, \ell_4, \ell_5\}$, 0) $\land$ Chk($\ell_0$, 1) $\land$ Chk($\ell_1$, 2) $\rightarrow$ Move($\ell_2$)
Rule20: Buddy = $\ell_4 \land$ Chk($\{\ell_1, \ell_2, \ell_3\}$, 0) $\land$ Chk($\ell_4$, 1) $\land$ Chk($\ell_5$, 2) $\rightarrow$ Move($\ell_4$)
Rule21: Buddy = $\ell_6 \land$ Chk($\{\ell_1, \ell_2, \ell_3\}$, 0) $\land$ Chk($\ell_4$, 1) $\land$ Chk($\ell_5$, 2) $\rightarrow$ Move($\ell_6$)
Rule22: Buddy = $\ell_0 \land$ Chk($\{\ell_1, \ell_4, \ell_5\}$, 0) $\land$ Chk($\ell_0$, 1) $\land$ Chk($\ell_2$, 1) $\rightarrow$ Move($\ell_1$)
Rule23: Buddy = $\ell_2 \land$ Chk($\{\ell_1, \ell_4, \ell_5\}$, 0) $\land$ Chk($\ell_0$, 1) $\land$ Chk($\ell_3$, 1) $\rightarrow$ Move($\ell_2$)
Rule24: Buddy = $\ell_4 \land$ Chk($\{\ell_1, \ell_4, \ell_5\}$, 0) $\land$ Chk($\ell_0$, 1) $\land$ Chk($\ell_3$, 1) $\rightarrow$ Move($\ell_3$)
Rule25: Buddy = $\ell_4 \land$ Chk($\{\ell_1, \ell_4, \ell_5\}$, 0) $\land$ Chk($\ell_0$, 1) $\land$ Chk($\ell_4$, 1) $\rightarrow$ Move($\ell_4$)
Figure 27: The proposed algorithm for the 7-pairbots-gathering
Rule26: Buddy = l₅ ∧ Chk({l₁, l₂, l₃, l₄}, 0) ∧ Chk({l₀, l₅, l₆}, 1) → Move(l₅)
Rule27: Buddy = l₆ ∧ Chk({l₂, l₃, l₄, l₅}, 0) ∧ Chk({l₀, l₁, l₆}, 1) → Move(l₆)
Rule28: Buddy = l₀ ∧ Chk({l₃, l₄, l₆}, 0) ∧ Chk(l₁, 1) ∧ Chk(l₃₃, 2) → Move(l₂)
Rule29: Buddy = l₀ ∧ Chk({l₁, l₄, l₅}, 0) ∧ Chk(l₃, 1) ∧ Chk(l₂, 2) → Move(l₃)
Rule30: Buddy = l₀ ∧ Chk({l₁, l₂, l₃}, 0) ∧ Chk(l₄, 1) ∧ Chk(l₂, 2) → Move(l₄)
Rule31: Buddy = l₀ ∧ Chk({l₁, l₂, l₄}, 0) ∧ Chk(l₅, 1) ∧ Chk(l₂, 2) → Move(l₅)
Rule32: Buddy = l₀ ∧ Chk({l₁, l₂, l₃}, 0) ∧ Chk(l₆, 1) ∧ Chk(l₂, 2) → Move(l₆)
Rule33: Buddy = l₀ ∧ Chk({l₂, l₃, l₄}, 0) ∧ Chk(l₁₃, 1) ∧ Chk(l₀, 2) → Move(l₁)
Rule34: Buddy = l₀ ∧ Chk({l₃, l₄, l₅, l₆}, 0) ∧ Chk(l₂, 1) ∧ Chk(l₁, 2) → Move(l₂)
Rule35: Buddy = l₁ ∧ Chk({l₂, l₃}, 0) ∧ Chk(l₁, 1) ∧ Chk(l₀, 2) → Move(l₁)
Rule36: Buddy = l₂ ∧ Chk({l₃, l₄}, 0) ∧ Chk(l₂, 1) ∧ Chk(l₀, 2) → Move(l₂)
Rule37: Buddy = l₆ ∧ Chk({l₁, l₂}, 0) ∧ Chk(l₆, 1) ∧ Chk(l₀, 2) → Move(l₆)
Rule38: Buddy = l₃ ∧ Chk({l₄, l₅}, 0) ∧ Chk(l₃, 1) ∧ Chk(l₀, 2) → Move(l₃)
Rule39: Buddy = l₄ ∧ Chk({l₅, l₆}, 0) ∧ Chk(l₄, 1) ∧ Chk(l₀, 2) → Move(l₄)
Rule40: Buddy = l₅ ∧ Chk({l₆, 0, l₀, l₁, l₄, l₅, l₆}, 0) ∧ Chk(l₅, 1) ∧ Chk(l₀, 2) → Move(l₅)
Rule41: Buddy = l₀ ∧ Chk({l₁, l₃, l₄, l₅, l₆}, 0) ∧ Chk(l₀, 1) ∧ Chk(l₆, 1) → Move(l₀)
Rule42: Buddy = l₀ ∧ Chk({l₂, l₃, l₄, l₅, l₆}, 0) ∧ Chk(l₃, 1) ∧ Chk(l₁, 2) → Move(l₃)
Rule43: Buddy = l₀ ∧ Chk({l₁, l₃, l₄, l₅, l₆}, 0) ∧ Chk(l₄, 1) ∧ Chk(l₀, 2) → Move(l₄)
Rule44: Buddy = l₀ ∧ Chk({l₁, l₃, l₄, l₅, l₆}, 0) ∧ Chk(l₂, 1) → Move(l₀)
Rule45: Buddy = l₁ ∧ Chk({l₂, l₀, l₁, l₃, 1} ∧ Chk({l₅, l₆, l₀, 2} → Move(l₁)
Rule46: Buddy = l₀ ∧ Chk({l₂, l₃, l₄, l₅, l₆}, 0) ∧ Chk(l₄, 1) ∧ Chk(l₃₃, 2) → Move(l₁)
Rule47: Buddy = l₄ ∧ Chk{l₃, 0} ∧ Chk(l₃₃, 1) ∧ Chk({l₀, l₁, l₂, l₃, 2} → Move(l₄)
Rule48: Buddy = l₃ ∧ Chk{l₄, 0} ∧ Chk(l₄₃, 1) ∧ Chk({l₀, l₁, l₃, l₄, 0} → Move(l₃)
Rule49: Buddy = l₂ ∧ Chk{l₅, l₆, 0} ∧ Chk(l₀, 1) → Move(l₂)
Rule50: Buddy = l₁ ∧ Chk{l₂, l₃, l₄, l₅, l₆, 0} ∧ Chk(l₀, l₁, 1) → Move(l₁)
Rule51: Buddy = l₄ ∧ Chk{l₅, 0} ∧ Chk{l₀, l₁, l₄, l₅, 1} ∧ Chk{l₃, 2} → Move(l₄)
Rule52: Buddy = l₄ ∧ Chk{l₅, 0} ∧ Chk{l₀, l₁, l₂, l₃, 2} → Move(l₄)
Rule53: Buddy = l₃ ∧ Chk{l₄, 0} ∧ Chk{l₀, l₂, l₃, l₄, 1} ∧ Chk{l₀, l₁, l₃, 2} → Move(l₃)
Rule54: Buddy = l₅ ∧ Chk{l₆, 0} ∧ Chk{l₁, l₄, 1} ∧ Chk{l₀, l₂, l₃, l₄, 2} → Move(l₅)
Rule55: Buddy = l₀ ∧ Chk{l₃, l₄, 0} ∧ Chk{l₁, l₄, l₆, l₀} ∧ Chk{l₅, 2} → Move(l₅)
Rule56: Buddy = l₁ ∧ Chk{l₄, 0} ∧ Chk{l₅, 1} ∧ Chk{l₀, l₃, l₄, l₅, 2} → Move(l₁)
Rule57: Buddy = l₃ ∧ Chk{l₅, 0} ∧ Chk{l₀, l₁, l₃, l₄, 1} ∧ Chk{l₃, 2} → Move(l₃)
Rule58: Buddy = l₅ ∧ Chk{l₆, 0} ∧ Chk{l₁, l₅, l₆, 1} ∧ Chk{l₀, l₃, l₄, 2} → Move(l₅)
Rule59: Buddy = l₃ ∧ Chk{l₅, l₆, 0} ∧ Chk{l₀, l₃, l₄, l₅, 1} ∧ Chk{l₃, 2} → Move(l₃)
Rule60: Buddy = l₄ ∧ Chk{l₆, 0} ∧ Chk{l₁, l₃, l₄, l₅, 1} ∧ Chk{l₃, 2} → Move(l₄)
Rule61: Buddy = l₆ ∧ Chk{l₁, l₂, l₃, l₄, 1} ∨ Chk{l₀, l₃, l₄, l₅, 2} → Move(l₆)
Rule62: Buddy = l₃ ∧ Chk{l₄, 0} ∧ Chk{l₀, l₃, l₄, l₅, 1} ∧ Chk{l₁, l₂, l₃, l₄, 2} → Move(l₃)
Rule63: Buddy = l₆ ∧ Chk{l₃, l₄, 0} ∧ Chk{l₁, l₃, l₄, 1} ∧ Chk{l₀, l₃, 2} → Move(l₆)
Rule64: Buddy = l₃ ∧ Chk{l₅, l₆, 0} ∧ Chk{l₁, l₃, l₄, l₅, 1} ∧ Chk{l₀, l₃, l₄, 2} → Move(l₃)
Rule65: Buddy = l₀ ∧ Chk{l₄, l₅, l₆, 0} ∧ Chk{l₁, l₃, l₄, l₅, 1} ∧ Chk{l₀, l₃, 2} → Move(l₀)
Rule66: Buddy = l₉ ∧ Chk{l₅, l₆, 0} ∧ Chk{l₁, l₃, l₄, l₅, 1} ∧ Chk{l₀, l₃, l₄, 2} → Move(l₉)
Rule67: Buddy = l₉ ∧ Chk{l₅, l₆, 0} ∧ Chk{l₁, l₃, l₄, l₅, 1} ∧ Chk{l₀, l₃, l₄, 2} → Move(l₉)
Rule68: Buddy = l₂ ∧ Chk{l₅, l₆, 0} ∧ Chk{l₁, l₃, l₄, l₅, 1} ∧ Chk{l₀, l₃, l₄, 2} → Move(l₂)
Rule69: Buddy = l₆ ∧ Chk{l₁, l₂, l₄, 0} ∧ Chk{l₀, l₃, l₄, l₅, 1} ∧ Chk{l₃, 2} → Move(l₆)
4.4 Correctness of the Proposed Algorithm

The proof of the correctness of the proposed algorithm is very challenging because it includes many rules. However, in this problem, the number of pairbot is fixed to 7, all pairbots are initially in short state, and only connected initial configurations are allowed; this implies that there exist only the constant number of initial configurations. Therefore, we implement a simulator (provided at [Kim and Taguchi [2023]]) for the proposed algorithm which can generate all possible initial configurations; there exist 3,652 initial configurations (as a result from the simulator). Note that the proposed algorithm is deterministic and we assume an FSYNC scheduler, thus there can be only one execution appears when an initial configuration is given. We had checked the proposed algorithm in every initial configuration using the simulation, and the proposed algorithm solves the 7-pairbots-gathering problem from all possible initial configurations. Therefore, the following theorem holds.

Theorem 6 The proposed algorithm solves the 7-pairbots-gathering problem from any arbitrary connected configuration under an FSYNC scheduler.

5 Conclusion

In this study, we have introduced a new computational model, the Pairbot model, consisting of paired robots that are called pairbots, which is based on the LCM model [Suzuki and Yamashita [1999]]. In the proposed model, each pairbot repeatedly changes the positional relation of the two robots (i.e., long and short) to achieve the goal. We presented the perpetual marching and the 7-pairbots-gathering problems to help to understand the computational power of the Pairbot model. In particular, from the perpetual marching problem, we can clarify the difference of the computational power in terms of a scheduler; the problem is solvable by 3 pairbots under an ASYNC scheduler, but unsolvable by 6 or less LCM-robots even under an SSYNC scheduler. Moreover, in the 7-pairbots-gathering problem, we showed the difference of the computational power in terms of a visibility range; the problem is solvable by pairbots with visibility range 1, but unsolvable by LCM-robots with the same visibility range (visibility range 2 is necessary to solve).

The Pairbot model basically has similar variations of assumptions (e.g., scheduler, geometric agreement, visibility, etc.), but it has one big different feature: every robot has one implicitly distinguishable robot as its buddy. We introduced only two problems here, but we are considering various problems in the Pairbot model such as pattern formation (a line, a triangle, or a hexagon), filling problem, and uniform deployment, as the future work. Especially, we are interested in the problems which have been solved in other computational models such as Amoebot, SILBOT, and MOBLOT to clarify the difference between Pairbot model and these computational models for PM.

The pairbot and the conventional LCM-models have many common features, thus, we can consider the simulation of the Pairbot model using the LCM model with some additional capabilities (e.g., light [Das et al. [2015]]). Clarifying the minimum required capabilities for the LCM model to simulate the Pairbot model is another future work.

References

Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

Shantanu Das, Paola Flocchini, Giuseppe Prencipe, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theoretical Computer Science*, 609, 09 2015.

Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors. *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*. Springer, 2019. ISBN 978-3-030-11071-0.

Giuseppe Prencipe. Autonomous mobile robots: A distributed computing perspective. volume 8243, pages 6–21, 12 2014.

G. Cicerone, Di Stefano, and A. Navarra. Gathering of robots on meeting-points. *Distributed Computing*, 31(1):1–50, 2018.

M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.

Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407:412–447, 11 2008.

Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, Peter Widmayer, Alok Aggarwal, and C. Rangan. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. volume 1741, pages 93–102, 01 1999.
N. Fujinaga, Y. Yamauchi, H. Ono, S. Kijima, and M. Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM Journal on Computing*, 44(3):740–785, 2015.

Y. Yamauchi, T. Uehara, S. Kijima, and M. Yamashita. Plane formation by synchronous mobile robots in the three-dimensional euclidean space. *Journal of the ACM*, 64(3):1:16:1–16:43, 2017.

V. Gervasi and G. Prencipe. Coordination without communication: The case of the flocking problem. *Discrete Applied Mathematics*, 144(3):324–344, 2004.

S. Souissi, T. Izumi, and K. Wada. Oracle-based flocking of mobile robots in crash-recovery model. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 683–697, 2009.

Taisuke Izumi, Samia Souissi, Yoshiaki Katayama, Nobuhiro Inuzuka, Xavier Défago, Koichi Wada, and Masafumi Yamashita. The gathering problem for two oblivious robots with unreliable compasses. *SIAM Journal on Computing*, 41, 11 2011.

Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Efficient dispersion of mobile robots on dynamic graphs. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 732–742, 2020. doi:10.1109/ICDCS47774.2020.00100

Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses. Byzantine dispersion on graphs. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 942–951, 2021. doi:10.1109/IPDPS49936.2021.00103

John Augustine and William K. Moses. Dispersion of mobile robots: A study of memory-time trade-offs. In *Proceedings of the 19th International Conference on Distributed Computing and Networking, ICDCN ’18*, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450363723. doi:10.1145/3154273.3154293 URL https://doi.org/10.1145/3154273.3154293

Lali Barriere, Paola Flocchini, Eduardo Mesa-Barrameda, and Nicola Santoro. Uniform scattering of autonomous mobile robots in a grid. In *2009 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, 2009. doi:10.1109/IPSITA.2009.5160871

Aisha Aljohani, Pavan Poudel, and Gokarna Sharma. Complete visitability for autonomous robots on graphs. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 733–742, 2018. doi:10.1109/IPDPS.2018.00083

Yonghwan Kim, Yoshiaki Katayama, and Koichi Wada. Asynchronous complete visibility algorithm for luminous robots on grid. In *Eleventh International Symposium on Computing and Networking, CANDAR 2023 - Workshops, Matsue, Japan, November 27-30, 2023*, pages 107–113. IEEE, 2023. doi:10.1109/CANDARW60564.2023.00026 URL https://doi.org/10.1109/CANDARW60564.2023.00026

Kevin Buchin, Paola Flocchini, Irina Kostitsyna, Tom Peters, Nicola Santoro, and Koichi Wada. Autonomous mobile robots: Refining the computational landscape. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 576–585, 2021. doi:10.1109/IPDPSW52791.2021.00091

Tommaso Toffoli and Norman Margolus. Programmable matter: Concepts and realization. *Int. J. High Speed Comput.*, 5(2):155–170, 1993.

Seth Copen Goldstein, Jason D. Campbell, and Todd C. Mowry. Programmable matter. *IEEE Computer*, 38(6):99–101, Jun 2005.

Josh C. Bongard. Evolutionary robotics. *Communications of ACM*, 56(8):74–83, 2013. ISSN 0001-0782.

Sangbae Kim, Cecilia Laschi, and Barry Trimmer. Soft robotics: a bioinspired evolution in robotics. *Trends in Biotechnology*, 31(5):287–294, 2013. ISSN 0167-7799.

Gianfranco Cerofolini, Paolo Amato, Massimo Masserini, and Giancarlo Mauri. A surveillance system for early-stage diagnosis of endogenous diseases by swarms of nanobots. *Advanced Science Letters*, 3:345–352, 12 2010. doi:10.1166/asl.2010.1138

A. L. Yarin. Nanofibers, nanofluidics, nanoparticles and nanobots for drug and protein delivery systems. *Scientia Pharmaceutica*, 78(3):542–542, 2010. ISSN 2218-0532.

Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: Amoebot—a new model for programmable matter. *Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 06 2014.
Joshua Daymude, Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa Richa, Christian Scheideler, and Thim Strothmann. On the runtime of universal coating for programmable matter. *Natural Computing*, 17, 11 2017a.

Rida A. Bazzi and Joseph L. Briones. Stationary and deterministic leader election in self-organizing particle systems. In Mohsen Ghaffari, Mikhail Nesterenko, Sébastien Tixeuil, Sara Tucci, and Yukiko Yamauchi, editors, *Stabilization, Safety, and Security of Distributed Systems - 21st International Symposium, SSS 2019, Pisa, Italy, October 22-25, 2019, Proceedings*, volume 11914 of *Lecture Notes in Computer Science*, pages 22–37. Springer, 2019.

Joshua J. Daymude, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Improved leader election for self-organizing programmable matter. In Antonio Fernández Anta, Tomasz Jurdzinski, Miguel A. Mosteiro, and Yanyong Zhang, editors, *Algorithms for Sensor Systems - 13th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALOSENSORS 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers*, volume 10718 of *Lecture Notes in Computer Science*, pages 127–140. Springer, 2017b.

Joshua Daymude, Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Christian Scheideler, and Andréa Richa. Convex hull formation for programmable matter. pages 1–10, 01 2020.

Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The canonical amoebot model: Algorithms and concurrency control. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

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.

Serafino Cicerone, Alessia Di Fonso, Gabriele Di Stefano, and Alfredo Navarra. MOBLOT: molecular oblivious robots. In Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé, editors, *AAMAS ’21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, pages 350–358. ACM, 2021.

Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Shape formation by programmable particles. *Distributed Comput.*, 33(1):69–101, 2020.

Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In Faramarz Fekri, Sasitharan Balasubramaniam, Tommaso Melodia, Ahmad Beirami, and Albert Cabellos, editors, *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication, NANOCOM’ 15, Boston, MA, USA, September 21-22, 2015*, pages 21:1–21:2. ACM, 2015.

Masahiro Shibata, Masaki Ohyabu, Yuichi Sudo, Junya Nakamura, Yonghwan Kim, and Yoshiaki Katayama. Visibility-optimal gathering of seven autonomous mobile robots on triangular grids. *International Journal of Networking and Computing*, 12(1):2–25, 2022.

Yonghwan Kim and Yuya Taguchi. 7-pairbots-gathering algorithm simulator and verifier, 2023. URL https://github.com/chaika1015/7pairbot-gathering.