Collision-Free Pattern Formation*

Rachid Guerraoui[1] and Alexandre Maurer†[2]

1 **EPFL, Lausanne, Switzerland**
`rachid.guerraoui@epfl.ch`
2 **EPFL, Lausanne, Switzerland**
`alexandre.maurer@epfl.ch`

Abstract

Shoals of small fishes can change their collective shape and form a specific pattern. They do so efficiently (in parallel) and without collision.

In this paper, we study the analog problem of distributed pattern formation. A set of processes needs to move from a set of initial positions to a set of final positions. The processes are oblivious (no internal memory) and must preserve, at any time, a minimal distance between them.

A naive solution would be to move the processes one by one, but this would take too long. The difficulty here is to move the processes simultaneously in clearly delimited phases, no matter how unfavorable the initial configuration may be. We solve this by treating the problem "dimension by dimension": the processes first form 1D trails, then gather into a 2D shape (this technique can be generalized to higher dimensions).

We present an optimal algorithm which time complexity depends linearly on the radius of the smallest circle containing both initial and final positions. The algorithm is self-stabilizing, as the processes are oblivious and the initial positions are arbitrary.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Pattern formation, Collision, Landmarks

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2016.16

1 Introduction

Nature has always been a fertile source of inspiration for computing: evolutionary algorithms [2], neural networks [16] or cellular automata [22] are all based on natural phenomena.

One remarkable phenomenon is the ability of simple individuals (ants, bees, fishes ...) to form complex patterns without centralized control. For instance, shoals of fishes can form a massive compact structure [20], which can then deform itself to cross small holes and avoid predators. These individuals form complex patterns very efficiently by moving to the new positions in parallel. They do so without accident: they do not even touch each other. Reproducing this phenomenon has a particular interest in the era of autonomous mobile devices [18]. This could have many applications, such as the exploration of dangerous or impracticable zones [3] as well as medical nanorobots [19].

In this paper, we consider the problem of arbitrary pattern formation with landmarks [14]: a set of n processes, with a set of n arbitrary initial positions, must move to a set of n arbitrary final positions (*landmarks*) forming some desired pattern. The difficulty of this

* This work has been supported in part by the Swiss National Science Foundation, grant 200021_169588 / TARBDA.
† Contact author.


problem lies on the requirement to consider "voluminous" processes (namely processes do have a volume). More specifically, a minimal *distance D* between any two processes needs to be preserved at any time. This requirement is essential for any application where the volume of processes is not negligible. Besides, many types of devices are fragile (quadcopter drones, robots exploring Mars ...) and collisions could render them unusable.

Most existing works on pattern formation [21, 8, 10, 23] ignore collisions: each process is represented by a point, and can move independently of other processes. Several processes can thus occupy the exact same position. Pattern formation has indeed been considered without collisions [13, 14, 15], but two processes could be as close as possible to each other, as long as they do not occupy the exact same point in space. These approaches do not apply to the setting where the volume of processes imposes a minimal distance between any two processes. Some papers considered problems where processes have a certain volume, such as localization [6], gathering [5, 17, 4, 1], circle formation [7] or coating [9]. However, the more general problem of arbitrary pattern formation has never been studied for voluminous processes.

We study this problem in a general setting where processes are *oblivious* (they have no form of internal memory) and cannot communicate with each other by message passing or signals. The only form of "communication" lies in the ability of each process to see the position of other processes (we do not put restrictions on their visibility). A naive solution would be to move the processes far away from the final positions, then to have the processes move one by one to the desired positions. Clearly, such a solution would be extremely ineffective in terms of execution time for a large number of processes (ideally, the processes should all move together).

The difficulty here is to separate the moves in clearly delimited *phases*, that can be identified without ambiguity only by looking at the positions of the processes. Besides, this should be done while all processes move simultaneously *and* with the constraint of respecting a minimal distance at any time, no matter how the processes are arranged in the initial configuration (i.e., possibly in a very unfavorable way).

In this paper, we present a solution which execution time depends only linearly on the distance between initial and final positions, which we show is optimal in terms of time complexity. Our algorithm involves two major steps:

- First, we provide a sub-algorithm to form *trails* without collisions (Collision-Free Trail Formation, CFTF). A trail is a slice of pattern contained in a very tight band. The idea underlying this algorithm is for the processes to scatter, align orthogonally to the band and finally gather.
- Then, we provide an algorithm that uses the CFTF algorithm as a subroutine (Collision-Free Pattern Formation, CFPF). This algorithm slices the desired pattern in several trails, scatters the trails in the orthogonal direction, uses the CFTF algorithm to form the desired trails, and finally gathers the trails to obtain the desired pattern.

The main idea here is to treat the problem "dimension by dimension". For presentation simplicity, we consider the pattern formation problem in a 2D space, but the same principle could be repeated to reach higher dimensions : in the same way that we stack "trails" to form a 2D pattern, we could stack "slices" very similar to 2D patterns to form any 3D pattern – and so forth.

Besides, our resulting CFPF algorithm is *self-stabilizing* [11], as the processes are oblivious and the initial positions are totally arbitrary. Thus, any transient disruption simply brings the system to a new initial configuration.

We also prove that this algorithm has an optimal time complexity. We express the time complexity as a function of R: the radius of the smallest circle containing both initial and

final positions of the processes. This is in contrast to classical time complexity that is often expressed as an asymptotic function of n (the total number of processes), which is not sufficient here: the execution time is conditioned by the initial and final distances between processes. The time complexity of the naive solution where the processes move one by one is $O(R^3)$ (see Section 2.3). We show that the time complexity of the pattern formation problem without collision is only $\Theta(R)$, and that our algorithm matches this bound (i.e., a tight bound on the time complexity is $\Omega(R)$).

The rest of the paper is organized as follows. In Section 2, we describe the problem, the model and our time complexity criteria (we show that this complexity is $\Omega(R)$), then we give an overview of our solution. In Section 3, we present our Collision-Free Trail Formation (CFTF) algorithm and show its $O(R)$ complexity. In Section 3, we present our Collision-Free Pattern Formation (CFPF) algorithm and show its $O(R)$ complexity. We conclude in Section 5.

2 Preliminaries

2.1 Model

We consider a 2D space with a (x, y) coordinates system, and a set P of n processes. Each process is described by its (x, y) position. We denote by $d(A, B)$ (resp. $d(p, q)$) the euclidean distance between two points A and B (resp. two processes p and q) in the space[1]. The processes must observe a minimal distance D between each other: for any two processes p and q, we must have $d(p, q) \geq D$.

We adopt the classical FSYNC (*fully synchronous*) model for swarm computing [12]. The time is divided into successive steps $t = 0, 1, 2, 3 \ldots$ Let $\epsilon > 0$ be an arbitrarily small constant. ϵ is the largest distance that a process can cross between two time steps. At each time t, each process p uses its position and the position of other processes to compute a point M such that $d(p, M) \leq \epsilon$, and moves to this point ("Look–Compute–Move"). Then, the position of p at time $t + 1$ will be M.

The processes are anonymous (one can distinguish two processes only by their position) and oblivious (no form of memory is available).

Note 1: In this problem, the processes are required to know n positions to reach in the plane (also called *landmarks* [14]). Thus, a common coordinates system is a legitimate hypothesis, as the processes can also know three specific points of the plane that provide origin, metric and orientation. In practice, as each process is assumed to see other processes, these three points could also be obtained from elements of the setting.

Note 2: We adopt a synchronous model for the clarity of presentation. It should not be seen as a succession of discrete steps where the processes "teleport" between each step, but rather as the approximation of a continuous move: the maximal distance ϵ that a process can cross between two steps can be as small as we want. In this paper, we focus on avoiding collisions during the pattern formation – which remains nontrivial even if we abstract symmetry or synchrony problems[2]

[1] If A (resp. B) has the coordinates (x, y) (resp. (x', y')), then $d(A, B) = \sqrt{(x - x')^2 + (y - y')^2}$.

[2] In [15], it was shown that a pattern formation problem solvable in the synchronous model was also solvable in the asynchronous model. However, it does not say anything about preserving a minimal distance between processes during the whole execution.

2.2 Problem

Let S be an arbitrary set of n points such that, for any two points A and B of S, $d(A, B) \geq D$. At $t = 0$, the n processes have an arbitrary position such that, for any two processes p and q of P, $d(p, q) \geq D$. The problem consists of finding an algorithm satisfying the two following conditions:

- **Liveness.** The n processes always eventually occupy the n positions of S permanently (and then do not move from these positions).
- **Safety.** At any point in time, the condition on the minimal distance between processes must always be respected: at any time and for any two processes p and q, $d(p, q) \geq D$.

For simplicity reasons, we only consider collisions at each discrete step. However, this is not a problem in practice, as ϵ can be as small as we want. To ensure that there is no collision between two steps, a simple solution is to impose a minimal distance $D + \epsilon$ instead of D. Also, note that even without this hypothesis, our algorithm prevents any collision between two steps (assuming that the processes move in straight line).

2.3 Time complexity

The performance of a pattern formation algorithm can be measured by its *time complexity*, that is: the asymptotic behavior of its execution time, i.e. the number of time steps required to form the pattern.

The time complexity of an algorithm is often expressed as a function of n (the total number of processes). However, this criteria is not sufficient in the case of mobile processes with a bounded speed: the distance between the initial and final positions imposes a minimal execution time, independently of n. In other words, it is impossible to bound the execution time with a function of n: if such a bound existed, it would always be possible to overstep it by increasing the distance between initial and final positions sufficiently.

Therefore, we consider another parameter here: R, the radius of the smallest circle that contains all initial and final positions of the processes. More precisely, R is the smallest number for which there exists a point O such that:

1. At $t = 0$, for any process p, $d(O, p) \leq R$.
2. For any point M of S, $d(O, M) \leq R$.

Note that, as there is a minimal distance between processes, R implicitly imposes a bound on n: the maximal number of processes that we can put into a circle of radius R with a minimal distance D between processes. In other words, the number of processes n can be proportional to R^2.

We want to express the time complexity T of the problem (that is, the number of steps to form the pattern) as a function of R. For this purpose, we use the classical asymptotic Landau notation: Ω, O and Θ.

- T is "$\Omega(f(R))$" if there exists R_0 and $k > 0$ such that $\forall R \geq R_0$, $T \geq kf(R)$. This is a *lower* bound: the time complexity is *at least* $kf(R)$ for a sufficiently large R.
- T is "$O(f(R))$" if there exists R_0 and $k > 0$ such that $\forall R \geq R_0$, $T \leq kf(R)$. This is an *upper* bound: the time complexity is *at most* $kf(R)$ for a sufficiently large R.
- T is "$\Theta(f(R))$" if T is both $\Omega(f(R))$ and $O(f(R))$ (*tight* bound). In other words, there exists R_0, $k_1 > 0$ and $k_2 > 0$ such that $\forall R \geq R_0$, $k_1 f(R) \leq T \leq k_2 f(R)$.

The time complexity of an algorithm where the processes move "one by one" (like in [4]) is at least proportional to nR. Thus, as n can be proportional to R^2, such an algorithm only

provides an upper bound $O(R^3)$ to the time complexity. In this paper, we show that the time complexity of the CFPF problem is only $\Theta(R)$:

- The lower bound is trivial. Consider the case where the n processes are on a circle of radius R, and that the center of this circle is one of the final positions. Then, the execution time is at least R/ϵ (the minimal time for a process to reach the center of the circle). Thus, the time complexity of the problem is $\Omega(R)$.
- For the upper bound, we provide an algorithm solving the CFPF problem with an execution time $O(R)$. We present this algorithm in the next section.

2.4 Overview of our algorithm

We give here an intuition of how we construct an algorithm to solve the CFPF problem in $O(R)$ time steps.

We first define the notion of a *trail*. Intuitively, a trail is a set of positions which is arbitrarily large in the x direction, but very tight in the y direction.

▶ **Definition 1** (Trail). For any point or process A, let $x(A)$ (resp. $y(A)$) be the x (resp. y) coordinate of A. A *trail* is a set of points T such that:

1. For any two points A and B of T, $d(A, B) \geq D$.
2. The "width" of T in the y direction is at most $D/2$: for any two points A and B of T, $|y(A) - y(B)| \leq D/2$.

Then, the set S of final positions can be seen as several trails stacked in the y direction, as shown below.

Let y_1 (resp. y_2) be the smallest (resp. largest) y coordinate occupied by a point of S. Let N be the smallest integer such that $y_1 + ND/2 > y_2$. $\forall i \in \{1, \ldots, N\}$, let T_i be the set of points M of S such that $y_1 + (i-1)D/2 \leq y(M) < y_1 + iD/2$. According to Definition 1, T_i is a trail. Then, the set S of final positions can be decomposed in the set of trails $\{T_1, \ldots, T_N\}$.

In Section 3, we give an algorithm to form any trail (Collision-Free Trail Formation algorithm), and bound its execution time. Then, in Section 4, we use this algorithm as a subroutine to define a CFPF algorithm with a $O(R)$ time complexity. This algorithm forms the trails $\{T_1, \ldots, T_N\}$, then stacks them in the y direction.

3 Collision-Free Trail Formation (CFTF) Algorithm

In this section, we give a CFTF algorithm and bound its execution time. We first define the CFTF problem, introduce some definitions and primitives, and give an informal description of the algorithm. Then, we describe the algorithm and show its execution time.

3.1 CFTF Problem

Consider a set Q of m processes[3]. Let T be a *trail* (see Definition 1) such that $|T| = m$. At $t = 0$, the m processes have an arbitrary position such that, for any two processes p and q of Q, $d(p, q) \geq D$. We want to find an algorithm such that the m processes always eventually occupy the m positions of T (*liveness*). During the whole process, the condition on the minimal distance between processes must always be respected: at any time and for any two processes p and q, $d(p, q) \geq D$ (*safety*).

[3] We use m instead of n to make a clear distinction between the CFTF problem and the CFPF problem.

3.2 Definitions

In order to describe our CFTF algorithm, we define the minimal x coordinate of the processes (resp. positions), define a total order relationship on the processes (resp. positions), and finally the elementary distances between processes (resp. positions) according to this order.

Minimal x position. Let $x_{\min}$ be the largest number such that, for each process p, $x(p) \geq x_{\min}$. In other words, $x_{\min}$ is the smallest x coordinate occupied by a process. Similarly, let x_0 be the smallest x coordinate occupied by a point of T.

Total order relationship. We define a total order relationship "$>$" on the position of processes. We say that $p > q$ if one of the two following conditions is satisfied:
(1) $x(p) > x(q)$,
(2) $x(p) = x(q)$ and $y(p) > y(q)$.

Note that the order relationship is total, as the condition on the minimal distance between processes forbids two processes to be at the same position. We also define a similar order relationship on the points of T.

For any time t, let $(p_1, p_2, \ldots, p_m)$ be the m processes such that $p_1 < p_2 < \cdots < p_m$. Let $(M_1, M_2, \ldots, M_m)$ be the m points of T such that $M_1 < M_2 < \cdots < M_m$.

Elementary x distances. $\forall i \in \{1, \ldots, m-1\}$, let $d_i = x(p_{i+1}) - x(p_i)$ (distances between processes) and $D_i = x(M_{i+1}) - x(M_i)$ (distances between positions).

3.3 Primitives

We define here some basic primitives used in the algorithm to describe the moves of the processes. Let p be a process with coordinates (x, y) at time t, and let M be a point of same coordinates.

"Move towards" primitive. Let M' be a point. Let K be a point of the segment $[MM']$ such that $d(M, K) = \epsilon$. If such a point does not exist (that is, if $d(M, M') < \epsilon$), we consider that $K = M'$. Then, we say that p *moves towards* M' if the position of p at time $t + 1$ is K.

"Move with vector" primitive. Let $[a, b]$ be a $2D$ vector such that $\sqrt{a^2 + b^2} \leq \epsilon$. Let K be a point of coordinates $(x + a, y + b)$. Then, we say that p *moves with vector* $[a, b]$ if the position of p at time $t + 1$ is K.

3.4 CFTF Algorithm

Our CFTF algorithm is described in Figure 1. We provide an informal description of the algorithm below.

The algorithm goes through 4 successive phases: the processes translate to the good x offset (Phase 1), increase their x distance between each other (Phase 2), align in the y direction (Phase 3) and finally adjust their distance in the x direction until they form the desired trail.

1. In Phase 1, the processes translate in the x direction until the smallest x coordinate occupied by a process ($x_{\min}$) corresponds to the smallest x coordinate of the set of final positions (x_0).

CFTF Algorithm

At each step, every process p executes the following algorithm, which is divided in 4 phases. The conditions of the 4 phases are defined to be mutually exclusive.

Predicates

- $P_1 : x_{\min} = x_0$
- $P_2 : \forall i \in \{1, \ldots, m-1\},\ d_i \geq \max(D, D_i)$.
- $P_3 : \forall i \in \{1, \ldots, m\},\ y(p_i) = y(M_i)$.
- P_4 : The m processes occupy the m positions of T.

Phase 1: Translation

Condition: $\neg P_1$

Action: If $|x_{\min} - x_0| < \epsilon$, move with vector $[x_0 - x_{\min}, 0]$. Otherwise:
- If $x_{\min} > x_0$, move with vector $[-\epsilon, 0]$.
- If $x_{\min} < x_0$, move with vector $[\epsilon, 0]$.

Phase 2: Scattering

Condition: $P_1 \wedge \neg P_2 \wedge \neg P_3$

Action: Let i be the smallest integer such that $d_i < \max(D, D_i)$.
Then, if p belongs to $\{p_{i+1}, p_{i+2}, \ldots, p_m\}$, move with vector $[\epsilon, 0]$.

Phase 3: Alignment

Condition: $P_1 \wedge P_2 \wedge \neg P_3$

Action: Let i be such that $p = p_i$. Let M be the point of coordinates $(x(p_i), y(M_i))$.
Then, move towards M.

Phase 4: Gathering

Condition: $P_1 \wedge P_3 \wedge \neg P_4$

Action: Let i be the smallest integer such that $d_i > D_i$. Let $e = d_i - D$.
Then, if p belongs to $\{p_{i+1}, p_{i+2}, \ldots, p_m\}$:
- If $e \leq \epsilon$, move with vector $[-e, 0]$.
- If $e > \epsilon$, move with vector $[-\epsilon, 0]$.

Figure 1 CFTF Algorithm.

2. In Phase 2, the processes scatter in the x direction until we have $d_i \geq \max(D, D_i)$ for each i. $d_i \geq D$ ensures that the processes can move in the y direction without collision in the next phase. $d_i \geq D_i$ ensures that the x distance between processes is at least the desired distance D_i. Thus, the processes only have to reduce this distance in Phase 4 to obtain the desired trail.
3. In Phase 3, the processes move in the y direction until they all have the desired y position. As we have $d_i \geq D$ for each i, there is no collision between processes during this phase.
4. In Phase 4, the processes gather in the x direction until we have $d_i = D_i$ for each i. According to Phase 2, they only have to reduce their distance between each other. Then, we have the desired x and y positions, and thus the desired trail.

3.5 Time complexity

Similarly to R (see Section 2.3), let r be the radius of the smallest circle containing all initial and final positions of the m processes. In Theorem 3, we show that the execution time of the CFTF algorithm is $O(r)$. For this purpose, we first bound m with r in Lemma 2.

▶ **Lemma 2.** $m \leq 1 + 2r/D$

Proof. First, let us show that $\forall i \in \{1, \ldots, m-1\}$, $D_i \geq D/2$. Indeed, assume the opposite: there exists i such that $D_i < D/2$. Then, there exists two final positions A and B such that $|x(A) - x(B)| \leq D/2$. As $|y(A) - y(B)| \leq D/2$, we have $d(A, B) \leq D/\sqrt{2} < D$: contradiction.

Therefore, $\Sigma_{i=1}^{i=m-1} D_i \geq (m-1)D/2$. As $\Sigma_{i=1}^{i=m-1} D_i \leq r$ (by definition), we have $(m-1)D/2 \leq r$. Thus, $m \leq 1 + 2r/D$. ◀

▶ **Theorem 3.** *The m processes occupy the m desired positions in $O(r)$ time steps, and the condition on the minimal distance is always respected.*

Proof. The proof is in 4 steps.

1. Suppose $\neg P_1$. The distance of translation in Phase 1 is at most r. Thus, P_1 is true after a time $T_1 \leq \lceil r/\epsilon \rceil = O(r)$. As the translation is the same for all processes, the safety condition is respected.
2. Suppose $P_1 \wedge \neg P_2 \wedge \neg P_3$. Let i be the integer described in Phase 2. Then, we have $d_i \geq \max(D, D_i)$ in at most $\lceil \max(D, D_i)/\epsilon \rceil$ time steps. Thus, $P_1 \wedge P_2 \wedge \neg P_3$ is true after a time $T_2 \leq \Sigma_{i=1}^{i=m-1} \lceil \max(D, D_i)/\epsilon \rceil \leq \Sigma_{i=1}^{i=m-1} \lceil D/\epsilon \rceil + \Sigma_{i=1}^{i=m-1} \lceil D_i/\epsilon \rceil \leq m(1 + D/\epsilon) + (m + \Sigma_{i=1}^{i=m-1} D_i/\epsilon) \leq (2 + D/\epsilon)m + r/\epsilon$. According to Lemma 2, $m \leq 1 + 2r/D$, and thus, $T_2 \leq 2 + D/\epsilon + (4/D + 3/\epsilon)r = O(r)$. As the moves of Phase 2 only increase the distance between processes, the safety condition is respected.
3. Suppose $P_1 \wedge P_2 \wedge \neg P_3$. The moves of Phase 1 and 2 do not modify the y position of the processes. Thus, $\forall i \in \{1, \ldots, m\}$, $|y(p_i) - y(M_i)| \leq r$. Therefore, in Phase 3, $P_1 \wedge P_3$ is true after a time $T_3 \leq \lceil r/\epsilon \rceil = O(r)$. According to P_2, for any two processes p and q, $|x(p) - x(q)| \geq D$. Thus, as the moves of Phase 3 are only in the y direction, the safety condition is respected.
4. Suppose $P_1 \wedge P_3 \wedge \neg P_4$. $\forall i \in \{1, \ldots, m-1\}$, Phase 2 increases d_i of at most $\max(D, D_i) + \epsilon$. Let Δ_i be the value of d_i at the beginning of Phase 4. Then, $\Sigma_{i=1}^{i=m-1} \Delta_i \leq r + \Sigma_{i=1}^{i=m-1} (\max(D, D_i) + \epsilon) \leq r + m(D + \epsilon) + \Sigma_{i=1}^{i=m-1} D_i \leq r + (1 + 2r/D)(D + \epsilon) + r \leq 2(2 + \epsilon/D)r + D + \epsilon$. Let i be the integer described in Phase 4. Then, we have $d_i = D_i$ in at most $\lceil \Delta_i/\epsilon \rceil \leq 1 + \Delta_i/\epsilon$ time steps. Thus, P_4 is true after a time $T_4 \leq \Sigma_{i=1}^{i=m-1} (1 + \Delta_i/\epsilon) \leq m + (\Sigma_{i=1}^{i=m-1} \Delta_i)/\epsilon \leq 1 + 2r/D + (2(2 + \epsilon/D)r + D + \epsilon)/\epsilon = 4(1/D + 1/\epsilon)r + 2 + D/\epsilon = O(r)$. As the moves of Phase 4 do not allow to have $d_i < D_i$, the safety condition is respected.

Once P_4 is true, the processes do not move, and thus the liveness condition is satisfied. The execution time is at most $T_1 + T_2 + T_3 + T_4 = O(r)$. ◀

4 Collision-Free Pattern Formation (CFPF) Algorithm

In this section, we use our previous CFTF algorithm to define a CFPF algorithm and bound its execution time. Similarly, we start with some definitions and an informal description of the algorithm.

4.1 Definitions

Similarly to the previous section, we first define the minimal y coordinate and a total order relationship now based on the y direction. We use this order to classify the processes in N sets $\{S_1, \dots, S_N\}$, which correspond to the N trails forming the final pattern. We then define the elementary distances between these sets and trails, and the y translation of a given trail.

- Similarly to Section 3.2, we define $y_{\min}$ as the smallest y coordinate occupied by a process. For a set of processes (resp. points) Q, let $y_{\min}(Q)$ be the smallest y coordinate occupied by a process (resp. point) of Q.
- The total order relationship defined in Section 3.2 gives priority to the x coordinate over the y coordinate. We now consider a total order relationship that gives priority to y over x. Let $(p_1, \dots, p_n)$ be the n processes ordered such that $p_1 < p_2 < \dots < p_n$, according to this order relationship. According to the definition of $\{T_1, \dots, T_N\}$ in Section 2.4, let S_1 be the $|T_1|$ first processes of the sequence, let S_2 be the $|T_2|$ following processes, and so forth until S_N.
- For two sets of processes $\{S, S'\} \subseteq \{S_1, \dots, S_N\}$, let $d(S, S') = \min_{(p,q)\in S\times S'} d(p, q)$. $\forall i \in \{1, \dots, N-1\}$, let $d_i = d(S_i, S_{i+1})$ (distance between sets) and $D_i = d(T_i, T_{i+1})$ (distance between trails).
- $\forall i \in \{1, \dots, N\}$ and for any value y_0, let $T'_i(y_0)$ be the trail obtained by a translation of T_i of distance $|y_0 - y_{\min}(T_i)|$ in the positive y direction.
- For a process p, let $S(p) \in \{S_1, \dots, S_N\}$ be the set such that $p \in S(p)$ (the set containing p).

4.2 CFPF Algorithm

The algorithm is described in Figure 2. We provide an informal description of the algorithm below.

Similarly to the CFTF algorithm of Section 3, the CFTF algorithm performs in 4 successive phases. These phases are defined so that the content of the sets $(S_1, \dots, S_N)$ always remains the same.

1. In Phase 1, the processes translate in the y direction until the smallest y coordinate of a process ($y_{\min}$) corresponds to the smallest y coordinate of the set of final positions ($y_{\min}(S)$).
2. In Phase 2, the sets $(S_1, \dots, S_N)$ scatter in the y direction until we have $d_i \geq \max(D, D_i)$ for each i. $d_i \geq D$ ensures that the processes can form the desired trails without collision in the next phase. $d_i \geq D_i$ ensures that the y distance between the sets is at least the desired distance D_i. Thus, the resulting trails only have to reduce this distance in Phase 4 to obtain the desired pattern.

CFPF Algorithm

At each step, every process p executes the following algorithm, which is divided in 4 phases. The conditions of the 4 phases are defined to be mutually exclusive.

Predicates

- P_1: $y_{\min} = y_{\min}(S_1)$
- P_2: $\forall i \in \{1, \ldots, N-1\}$, $d_i \geq \max(D, D_i)$.
- P_3: $\forall i \in \{1, \ldots, N\}$, there exists y_i such that $S_i = T'_i(y_i)$.
- P_4: The n processes occupy the n desired positions.

Phase 1: Translation

Condition: $\neg P_1$

Action: If $|y_{\min} - y_0| < \epsilon$, move with vector $[0, y_0 - y_{\min}]$. Otherwise:
- If $y_{\min} > y_0$, move with vector $[0, -\epsilon]$.
- If $y_{\min} < y_0$, move with vector $[0, \epsilon]$.

Phase 2: Scattering

Condition: $P_1 \wedge \neg P_2 \wedge \neg P_3$

Action: Let i be the smallest integer such that $d_i < \max(D, D_i)$.
Then, if $S(p)$ belongs to $\{S_{i+1}, S_{i+2}, \ldots, S_N\}$, move with vector $[0, \epsilon]$.

Phase 3: Trail formation

Condition: $P_1 \wedge P_2 \wedge \neg P_3$

Action: Let i be such that $S(p) = S_i$. Let $y_i = y_{\min}(S_i)$. Then, execute the CFTF algorithm of Section 3 to form the trail $T'_i(y_i)$ with the other processes of S_i.

Phase 4: Gathering

Condition: $P_1 \wedge P_3 \wedge \neg P_4$

Action: Let i be the smallest integer such that $d_i > D_i$. Let $e = d_i - D$.
Then, if $S(p)$ belongs to $\{S_{i+1}, S_{i+2}, \ldots, S_N\}$:
- If $e \leq \epsilon$, move with vector $[0, -e]$.
- If $e > \epsilon$, move with vector $[0, -\epsilon]$.

Figure 2 CFPF Algorithm.

3. In Phase 3, the processes of each set S_i use the CFTF algorithm of Section 3 to form a trail similar to T_i, but translated in the y direction.
4. In Phase 4, the sets $(S_1, \ldots, S_N)$ gather in the y direction until we have $d_i = D_i$ for each i. Then, the processes form the N desired trails, and thus the desired pattern.

Note that the CFTF algorithm does not increase (resp. decrease) the maximal (resp. minimal) y coordinate occupied by a process of the trail. Thus, the condition of Phase 3 always remains true during the execution of the CFTF subroutine.

4.3 Time complexity

▶ **Theorem 4.** *The n processes occupy the n desired positions in $O(R)$ time, and the condition on the minimal distance is always respected.*

Proof. The proof is in 4 steps.

1. Suppose $\neg P_1$. The distance of translation in Phase 1 is at most R. Thus, P_1 is true after a time $T_1 \leq \lceil R/\epsilon \rceil = O(R)$. As the translation is the same for all processes, the safety condition is respected.
2. Suppose $P_1 \wedge \neg P_2 \wedge \neg P_3$. Let i be the integer described in Phase 2. Then, we have $d_i \geq \max(D, D_i)$ in at most $\lceil \max(D, D_i)/\epsilon \rceil$ time steps. Thus, $P_1 \wedge P_2 \wedge \neg P_3$ is true after a time $T_2 \leq \Sigma_{i=1}^{i=N-1} \lceil \max(D, D_i)/\epsilon \rceil \leq \Sigma_{i=1}^{i=N-1} \lceil D/\epsilon \rceil + \Sigma_{i=1}^{i=N-1} \lceil D_i/\epsilon \rceil \leq N(1 + D/\epsilon) + (N + \Sigma_{i=1}^{i=N-1} D_i/\epsilon) \leq (2 + D/\epsilon)N + R/\epsilon$. According to Section 2.4, N is the smallest integer such that $y_1 + ND/2 > y_2$. Thus, $y_1 + (N-1)D/2 \leq y_2$, $(N-1)D/2 \leq y_2 - y_1 \leq R$, and $N \leq 1 + 2R/D$. Then, $T_2 \leq 2 + D/\epsilon + (4/D + 3/\epsilon)R = O(R)$. As the moves of Phase 2 only increase the distance between processes, the safety condition is respected.
3. Suppose $P_1 \wedge P_2 \wedge \neg P_3$. Let $i \in \{1, \ldots, N\}$. The moves of Phase 1 and 2 do not increase the distance between the processes of S_i. Therefore, all the processes of S_i are still contained in a circle of diameter R. Thus, as the processes of S_i execute the CFTF algorithm in Phase 3, according to Theorem 3, $P_1 \wedge P_3 \wedge \neg P_4$ is true after a time $T_3 = O(R)$. Let Y_i (resp. Y_i') be the smallest (resp. largest) y coordinate occupied by a process of S_i. During Phase 3, according to the CFTF algorithm, Y_i does not increase and Y_i' does not decrease. Thus, during Phase 3, $\forall i \in \{1, \ldots, N-1\}$, we have $d_i \geq D$, and the safety condition is respected.
4. Suppose $P_1 \wedge P_3 \wedge \neg P_4$. $\forall i \in \{1, \ldots, N-1\}$, Phase 2 increases d_i of at most $\max(D, D_i) + \epsilon$. Let Δ_i be the value of d_i at the beginning of Phase 4. Then, $\Sigma_{i=1}^{i=N-1} \Delta_i \leq R + \Sigma_{i=1}^{i=N-1} (\max(D, D_i) + \epsilon) \leq R + N(D + \epsilon) + \Sigma_{i=1}^{i=N-1} D_i \leq D + \epsilon + 2(2 + \epsilon/D)R$. Let i be the integer described in Phase 4. Then, we have $d_i = D_i$ in at most $\lceil \Delta_i/\epsilon \rceil \leq 1 + \Delta_i/\epsilon$ time steps. Thus, P_4 is true after a time $T_4 \leq \Sigma_{i=1}^{i=N-1} (1 + \Delta_i/\epsilon) \leq N + (\Sigma_{i=1}^{i=N-1} \Delta_i)/\epsilon \leq N + D/\epsilon + 1 + 2(2/\epsilon + 1/D)R = O(R)$. As the moves of Phase 4 do not allow to have $d_i < D_i$, the safety condition is respected.

Once P_4 is true, the processes do not move, and thus the liveness condition is satisfied. The execution time is at most $T_1 + T_2 + T_3 + T_4 = O(R)$. ◀

5 Conclusion

We gave and proved the first algorithm for pattern formation in the plane that always preserves a minimal distance between processes. We showed that the time complexity of this problem depends linearly on the diameter of the set of initial and final positions, and that our algorithm matches this bound.

A difficult open problem would be to study collision-free pattern formation in the presence of defective or malicious processes, that move independently of the algorithm and perturb the pattern formation. Also, an important step towards practical applications would be to study the impact of small errors and imprecision in the moves of processes and in their visibility mechanism. We also adopted a model where the processes have a global visibility of the swarm, and a challenging open problem would be to consider limited visibility.

References

1 Chrysovalandis Agathangelou, Chryssis Georgiou, and Marios Mavronicolas. A distributed algorithm for gathering many fat mobile robots in the plane. In *ACM Symposium on Principles of Distributed Computing, PODC'13, Montreal, QC, Canada, July 22-24, 2013*, pages 250–259, 2013. doi:10.1145/2484239.2484266.

2 Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

3 Leoncio Briones, Paul Bustamante, and Miguel Serna. Wall-climbing robot for inspection in nuclear power plants. In *Robotics and Automation*, pages 1409–1414. IEEE, 1994.

4 Sruti Gan Chaudhuri and Krishnendu Mukhopadhyaya. Gathering asynchronous transparent fat robots. In *Distributed Computing and Internet Technology*, pages 170–175. Springer, 2010.

5 Jurek Czyzowicz, Leszek Gasieniec, and Andrzej Pelc. Gathering few fat mobile robots in the plane. *Theoretical Computer Science*, 410(6):481–499, 2009.

6 Jurek Czyzowicz, Evangelos Kranakis, and Eduardo Pacheco. Localization for a system of colliding robots. In *Automata, Languages, and Programming*, pages 508–519. Springer, 2013.

7 Suparno Datta, Ayan Dutta, Sruti Gan Chaudhuri, and Krishnendu Mukhopadhyaya. Circle formation by asynchronous transparent fat robots. In *Distributed Computing and Internet Technology, 9th International Conference, ICDCIT 2013, Bhubaneswar, India, February 5-8, 2013. Proceedings*, pages 195–207, 2013. doi:10.1007/978-3-642-36071-8_15.

8 Xavier Défago and Samia Souissi. Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. *Theoretical Computer Science*, 396(1):97–112, 2008.

9 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, Thim Strothmann, and Shimrit Tzur-David. Infinite object coating in the amoebot model. *CoRR*, abs/1411.2356, 2014. URL: http://arxiv.org/abs/1411.2356.

10 Yoann Dieudonné and Franck Petit. Scatter of robots. *Parallel Processing Letters*, 19(01):175–184, 2009.

11 Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

12 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by oblivious mobile robots. *Synthesis Lectures on Distributed Computing Theory*, 3(2):1–185, 2012.

13 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1):412–447, 2008.

14 Nao Fujinaga, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. Pattern formation through optimum matching by oblivious CORDA robots. In *Principles of Distributed Systems – 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings*, pages 1–15, 2010. doi:10.1007/978-3-642-17653-1_1.

15 Nao Fujinaga, Yukiko Yamauchi, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM J. Comput.*, 44(3):740–785, 2015. doi:10.1137/140958682.

16 Martin T. Hagan, Howard B. Demuth, Mark H. Beale, et al. *Neural network design*. Pws Pub. Boston, 1996.

17 Anthony Honorat, Maria Potop-Butucaru, and Sébastien Tixeuil. Gathering fat mobile robots with slim omnidirectional cameras. *Theoretical Computer Science*, 557:1–27, 2014.

18 Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

19 Sylvain Martel, Mahmood Mohammadi, Ouajdi Felfoul, Zhao Lu, and Pierre Pouponneau. Flagellated magnetotactic bacteria as controlled mri-trackable propulsion and steering systems for medical nanorobots operating in the human microvasculature. *The International journal of robotics research*, 28(4):571–582, 2009.

20 Michael S. Mooring and Benjamin L. Hart. Animal grouping for protection from parasites: selfish herd and encounter-dilution effects. *Behaviour*, 123(3):173–193, 1992.

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

22 Stephen Wolfram et al. *Theory and applications of cellular automata*, volume 1. World Scientific Singapore, 1986.

23 Yukiko Yamauchi, Taichi Uehara, Shuji Kijima, and Masafumi Yamashita. Plane formation by synchronous mobile robots in the three dimensional euclidean space. In *Distributed Computing – 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*, pages 92–106, 2015. doi:10.1007/978-3-662-48653-5_7.