Synthesizing Algorithms to Avoid an Obstacle with a Swarm of Robots

Karine Altisen*, Anaïs Durand†, Pascal Lafourcade†, and Oussama Nahnah†
**Université de Genève*, Switzerland
karine.altisen@unige.ch
†*Université Clermont Auvergne, Clermont Auvergne INP, CNRS, LIMOS*, France
firstname.lastname@uca.fr

***Abstract*—Designing distributed algorithms for a swarm of robots and proving their soundness is a complex, highly combinatorial, and error-prone task. While some attempts at automation have been made, they have remained limited. We propose a partly automated method to synthesize algorithms that solve the perpetual exploration of a finite grid containing an obstacle using luminous myopic robots, with common chirality, in the fully-synchronous Look-Compute-Move model. This generic method is supported by a proof-of-concept tool that generates algorithms and verifies them by simulation on small grids. The results of the simulation are generalized to larger grids with a pen-and-paper proof.**

***Index Terms*—Luminous Robots, Perpetual Exploration, Finite Grid, Obstacle, Synthesis**

I. Introduction

Swarm robotics is an active research field these last years. Given a set of autonomous mobile entities called robots, the objective is to study the minimum hypotheses under which the robots can solve some complex tasks.

We consider *autonomous* mobile robots that are *luminous*, *i.e.*, they are endowed with lights of a finite number of colors, and evolve in a discrete environment modeled by a graph. They are not able to directly communicate but are equipped with visibility sensors allowing them to perceive their surrounding environment. They are *myopic* meaning that their visibility range is limited. We consider robots that operate under the *fully-synchronous Look-Compute-Move (LCM) model*, meaning that they execute synchronous rounds composed of three atomic phases: Look, Compute, and Move. During the Look phase, they take a snapshot of their environment within their visibility range. Then, they use the snapshot to decide their next destination and change or not their color in the Compute phase. Finally, they move to their next destination, if needed, in the Move phase.

Designing distributed algorithms for swarm of robots under those hypotheses is a complex and error-prone task and proving their correctness may be tedious since the problem is highly combinatorial with numerous assumption variants to handle. While the common practice is to design distributed algorithms by hand and to ensure their correctness by providing proofs on paper, computer-aided tools were proposed to improve those practices.

For example, using *proof-assistants* such as Rocq [1] (previously named *Coq*) allows to formalize the proofs of a given algorithm and certify their correctness [2]. While it allows strong guarantees by ensuring the correctness of the algorithm for every parameter value (*e.g.*, topology of the graph), this approach is not automatic. The proof designer must come with (at least) a sketch of the proof and the tool is used as a guidance to enumerate cases, avoid flaws, and clarify the assumptions. On the other hand, approaches based on *model-checking* [3], [4] or *controller synthesis* [5], [6] were proposed. These approaches require either to completely define the full context of execution (*e.g.*, number of robots, number of colors, graph topology) or to provide parametric solutions which are often restricted due to undecidability limitations [7]. Moreover, those methods usually face a combinatorial explosion with huge execution space, restricting their applicability, see for example the synthesis of correct by construction algorithms restricted to few robots and very small graphs [5], [6].

To circumvent those difficulties, we propose a different approach which combines automated and by-hand steps: our method guides the design by proposing patterns of moves for the swarm, automatically synthesizes algorithms, and then proves their correctness. To illustrate our method, we consider a classical problem of swarm robotics, under new settings: the *perpetual exploration of a finite grid with an obstacle*. In this problem, the robots must explore the grid by visiting each location of the grid infinitely often. While this problem has been extensively considered for luminous robots under the LCM model [8]–[10], none of the existing results considered grids with an obstacle.

A. Contributions

We propose a *partly automated method* to obtain new algorithms for a swarm of robots that perpetually explores a finite grid with one obstacle. The robots are luminous, myopic, disoriented (without compass) but with common chirality, cannot directly communicate, and execute in the fully-synchronous LCM model. By obstacle, we mean that the robots can encounter a pole or a hole in the grid, meaning that they have to bypass it unless bumping or falling into it.

The synthesis of algorithms that answer this problem quickly faces the state space explosion problem, limiting the approach to very small instances of the problem. To circumvent this and obtain solutions that satisfy the specification for

a parametric size of grids, we tackle the problem at several levels, as explained in the five steps of our method:

1) Instead of designing fully novel algorithms, we extend an existing algorithm which performs the perpetual exploration for grids (of certain sizes) with no obstacles.
2) We first synthesize new candidate algorithms: to limit the state space exploration, we guide the searching by designing a scenario that guides the robots when bypassing the obstacle.
3) Synthesized candidate algorithms may not satisfy the perpetual exploration: they are tested by simulations on small grids and only those which validate the property on those grids remain as the synthesized valid solutions. This step again may be overwhelming long and can be restricted by simulating a subset of the synthesized candidates; in our method we propose to use the energy consumption of each algorithm to test only the promising less greedy ones.
4) The fact that a valid solution satisfies the perpetual exploration of larger grids (than the tested ones) is then obtained by a general pen-and-paper proof. This proof is done by induction where the base case is made of the simulations on small grids and the induction step extends the grid by adding column(s) and/or line(s).
5) The method computes a set of algorithms that answer the problem. This set may contain a lot of elements and as every automatic (synthesis) tool, it brings no insight about the algorithms. This is why, as a last step of the method, we propose to analyse the results and classify the algorithms using the following criteria: number of rules, number of rounds required to perform a cycle of the exploration of the grid, energy consumption and paths taken by the robots during this exploration cycle.

The proposed method is *generic* w.r.t. the existing algorithm and supported by a *proof of concept tool* which automatizes Steps 2, 3, and 5 above. Its code is available at [11].

We apply our method on *two case studies*: we use two of the algorithms proposed in [9] which assume two disoriented myopic robots (with visibility one or two depending on the case study) with common chirality and we synthesize new algorithms that answer the problem. We obtain 5 (resp. 115) new algorithms for the first (resp. second) case study. Those algorithms are compared using the four criteria presented before, proving that they are all different solutions to the problem. We also manage to show the genericity of the method: by changing some parameters of the tool, new algorithms can be easily synthesized.

Roadmap: The rest of the paper is organized as follows. The next paragraph revisits related works in the light of our problem. In Section II, we present the computational model, illustrated by the algorithms used in our case study. In Section III, we define our method. We apply it to the case studies in Section IV which also introduces the proof of concept tool that supports the method. We conclude the paper in Section V.

B. Related Work

We first present related work about the problem of designing algorithms for a swarm of robots that in particular performs perpetual exploration; and then we focus on formal approaches that prove / verify / synthesize algorithms for such problems.

a) Perpetual exploration of a finite grid: Exploration of finite grids has been extensively studied in the luminous robots model. Several studies investigate the *terminating exploration* [12], [13], where robots must stop their execution once they have explored the grid. On the contrary, in the *perpetual exploration problem*, the robots must infinitely often explore each location of the grid. This problem has also been studied under various settings. In [8], the authors consider the perpetual exploration of a finite grid by myopic luminous robots that share a common chirality. The authors in [10] study the same problem without assuming chirality. The authors of [8] extend these results to the asynchronous model.

Some studies considered terminating exploration with obstacles [14], [15] but for oblivious (*i.e.,* without lights) non-myopic robots. Moreover, in [15] the authors allow robots to communicate using whiteboards on the nodes of the grid. However, to our knowledge, the perpetual exploration problem has never been studied in environments with obstacles.

b) Formal Approaches: Various computer-aided approaches have been considered for swarm robotics to verify properties or synthesize algorithms [16]. Verifying properties can be done using *proof assistants* such as *Rocq*. In particular, the Pactole framework [2] allows to certify impossibility results and soundness of protocols for swarms of robots using Rocq. Numerous swarm of robots algorithms have been certified using Pactole, *e.g.*, [17]–[20]. But in our knowledge, only terminating exploration has been considered [19].

Model-checking is another approach that has been used to verify whether a distributed robot algorithm satisfies its specification by exploring all possible system states [3], [4]. However, it usually faces the huge execution space problem, leading to limited results. In [3], the authors consider the perpetual exploration of rings using oblivious robots. They use model-checking to automatically verify an algorithm that requires three robots but only in rings of size n, with $10 \leq n \leq 16$. In [4], model-checking is used to verify an algorithm for the *rendez-vous* problem (also called *gathering*) in a 2D Euclidean space with luminous robots. In this problem, two robots must eventually reach the same location. To our knowledge, none of the model-checking approaches considered the perpetual exploration problem using luminous robots.

Finally, few studies considered *synthesizing* distributed algorithms for swarm of robots [5], [6], restricted to the ring topology. Similarly to model-checking, those studies are limited due to combinatorial explosion. In [6], the authors consider the gathering problem, where robots must meet at the same location. In this study, they synthesize algorithms for three oblivious robots in rings of size n, with $3 \leq n \leq 15$ and $n = 100$. The algorithm is then generalized by-hand to any ring size. In [5], the authors consider the perpetual exploration problem using oblivious robots in rings of size 10. They

automatically generate algorithm correct-by-construction with 3, 4, 6, or 7 robots. In this paper, rather than rings, we focus on grids and we consider luminous robots instead of oblivious ones. Under this context, the combinatorial explosion rises far earlier, due to the increased possibilities of communications for each robots.

II. Perpetual Exploration of a Grid

We consider a set of $n > 0$ *uniform autonomous luminous robots* [21] located on a finite rectangular grid of $\mathcal{C}$ columns and $\mathcal{L}$ lines represented by a non-oriented graph $\mathcal{G} = (V, E)$. The robots are placed on the nodes V of $\mathcal{G}$ and can move from one node to the other by traversing the edges E. The borders of the finite grid are called *walls*. The grid is *anonymous*, meaning that grid coordinates are used in the analysis of the algorithms but robots cannot access them (we denote a node coordinates by $(i, j) \in \{1, ..., \mathcal{C}\} \times \{1, ..., \mathcal{L}\}$).

Those robots are equipped with a fixed number of lights. They have no other memory – in particular they do not remember their previous actions – and no direct mean of communication. The robots are equipped with sensors to capture a *(local) view* of their surroundings, *i.e.*, the state of the neighboring nodes (empty node, wall, obstacle or color of an occupying robot). But they are *myopic* with a given *visibility range* $\phi \geq 1$ (see Fig 1).

The robots do not have a global orientation system or compass: their local view is not oriented. We consider only robots with *common chirality*. This means that the robots cannot distinguish a view from its $\frac{\pi}{2}$, π, and $\frac{3\pi}{2}$ rotations.

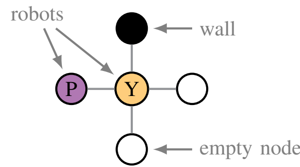

Fig. 1: Example of view of a robot (Y) with $\phi = 1$.

a) Algorithm: Each robot executes *fully-synchronous rounds* in the *Look-Compute-Move* (LCM) model [22]. In each round, every robot synchronously and atomically executes the three following phases. In the *Look* phase, the robot captures the snapshot of its local view. Then, the robot *computes* its next direction and changes its color (if needed) with respect to its algorithm. Finally, it *moves* towards its new position (or remains idle).

An *algorithm* $\mathcal{A}$ is a tuple $(Col, \mathcal{I}, Rules)$ where Col is a set of possible colors, $\mathcal{I}$ is a set of initial configurations in the grid, and $Rules$ is a set of rules. A *configuration* is a mapping between the nodes of $\mathcal{G}$ and their content at a given time (empty or with a robot of color $c \in Col$). A *rule* is a tuple (v, d, c), where v is a (local) view, $d \in \{Idle, Up, Down, Left, Right\}$ is a direction, and $c \in Col$, see for example Fig. 2. Rules where the robot remains idle and does not change its color are ignored.

During the Compute phase, the robot computes which rule to apply by comparing the snapshot of its local view (and its indistinguishable rotations) to the views of the rules, if any. In a *deterministic* algorithm at most one rule per robot applies at any round meaning that no pair of rules can have indistinguishable views (due to common chirality). Notice that robots are *self-inconsistent* meaning that they may not apply the same rotation at each round. The algorithm is *well-defined* if the destination of any robot (in the global grid) is entirely determined by the rule it executes despite its indistinguishable rotations. Finally, no collision between robots must happen, *i.e.*, two robots cannot be collocated on the same node and they cannot simultaneously cross the same edge to exchange their position. If the latter property is verified, the algorithm is said to be *exclusive*. An algorithm is *valid* if it has all those properties: deterministic, well-defined, and exclusive.

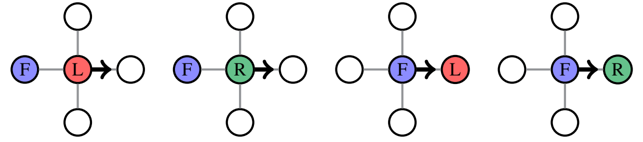

Fig. 2: Some rules of $\mathcal{A}_b^1$ [9].

b) Example of Algorithms: We consider the two algorithms for two robots given in [9]; those algorithms will be used as *basis* algorithms in the case studies, see Section IV. The first one called here $\mathcal{A}_b^1$, assumes visibility range one and uses three colors in $Col = \{L, R, F\}$; the second one, $\mathcal{A}_b^2$, assumes visibility range two and two colors L and F.

In $\mathcal{A}_b^1$, the two robots explore the grid in a serpentine way. One robot is the *leader* (with colors L or R) and the other is the *follower* (with color F). Both robots are always side-by-side.[1] The robots explore the grid row by row: the leader moves away from the follower and the follower follows (see some rules in Fig. 2). When the leader encounters a wall, it turns to the next row on its left if it is colored L, on its right otherwise, and switches color. $\mathcal{A}_b^2$ works similarly but instead of using two colors at the leader to distinguish whether it must turn left or right at the wall, the algorithm only uses L and exploits the distance between the leader and the follower. The leader turns left when the follower is at distance one, right when it is at distance 2. The reader can consult [23] and [24], the online animations of $\mathcal{A}_b^1$ and $\mathcal{A}_b^2$, respectively.

c) Execution: An *execution* $e = (\gamma_t)_{t \geq 0}$ is a sequence of configurations where γ_0 is the *initial configuration* and, for every $t \geq 1$, γ_t is the configuration reached from γ_{t-1} after executing a Look-Compute-Move round.

An algorithm has *locally-defined initial configurations* if its set of initial configurations is defined by colors and relative positions of the robots only. Ensuring the exploration starting from an arbitrary initial configuration is very difficult, maybe even impossible. Thus, designing algorithms with locally defined initial configurations is a nice compromise that allows more flexible initial configurations than a fixed one. In particular, $\mathcal{A}_b^1$ and $\mathcal{A}_b^2$ have locally defined initial configurations. Indeed, the set of initial configurations $\mathcal{I}_1$ for $\mathcal{A}_b^1$ is defined as any configurations where the two robots are side-by-side, one being of color F, the other of color L or R, anywhere in the

[1] To fix a convention on the vocabulary we say that the robots are in the same row but two consecutive columns.

grid. The initial configurations, $\mathcal{I}_2$, of $\mathcal{A}_b^2$ are those where the leader robot, with color L, and the follower one, with color F, are colinear and at distance 1 or 2.

Given a set of initial configurations, $\mathcal{I}$, the set of *reachable configurations*, $\mathcal{R}$, is the set of configurations that can be reached by an execution of the algorithm starting from an initial configuration. Notice that usually the set of initial configurations is simply included in the set of reachable ones but for $\mathcal{A}_b^1$, we have the particular property that $\mathcal{R}_1 = \mathcal{I}_1$.

A *local configuration* is the projection of a configuration γ on the nodes of the grid that are visible by at least one robot. This projection is devoided of global coordinates. Two different configurations γ and γ' can have the same projection: in this case, the robots behave the same way in γ and γ' since they cannot distinguish the local configurations.

d) Perpetual exploration: An execution $e = (\gamma_t)_{t\geq 0}$ in a grid $\mathcal{G}$ performs *perpetual finite grid exploration* if, for any node $v \in V$ and for any time $t \geq 0$, there exists a time $t' \geq t$ such that v is occupied by a robot in configuration $\gamma_{t'}$.

An algorithm $\mathcal{A}$ satisfies *perpetual finite grid exploration* for some grid $\mathcal{G}$, if given a set of initial configurations $\mathcal{I}$, every execution of $\mathcal{A}$ starting from $\mathcal{I}$ performs perpetual finite grid exploration of $\mathcal{G}$. In particular:

Theorem 1 ([9]). *For any grid $\mathcal{G}$ of size $\mathcal{C} \times \mathcal{L}$, $\mathcal{C}, \mathcal{L} \geq 2$, $\mathcal{A}_b^1$ (resp. $\mathcal{A}_b^2$) satisfies perpetual finite grid exploration starting from $\mathcal{I}_1$ (resp. $\mathcal{I}_2$).*

e) Obstacles: An *obstacle* is an element which is added at some node of the grid; it is called an obstacle since no robot is allowed to enter such a node. Usually robots have to bypass the obstacle in order to continue their trajectory.

We focus on adapting existing algorithms to perform perpetual exploration in grids that may contain a *single* obstacle. We consider two types of obstacles: either a see-through obstacle such as a *hole* (the robots can see what is behind the obstacle, if a large enough visibility range), or an opaque obstacle distinguishable from a wall such as a *pole* (the robots cannot see what is behind the obstacle). Obviously, algorithms $\mathcal{A}_b^1$ and $\mathcal{A}_b^2$ are no more adapted to such new types of nodes: since none of their rules takes into account the fact that a robot may encounter a hole or pole, the algorithms deadlock when it happens. But adding a new type of nodes such as a hole or pole opens possibilities to design new rules in order to bypass the obstacle and perform perpetual exploration of the grid.

Now, handling an obstacle near the border of the grid is difficult and can even be impossible. For example, a *dead-end obstacle*[2] would lead to collisions between robots with Algorithm $\mathcal{A}_b^1$, see our technical report [11], Section II.A for the proof. To avoid those situations, we consider an obstacle which is positioned sufficiently far from the grid borders so that the robots cannot see both the obstacle and the border simultaneously; namely we define the predicate $\mathit{ObsInTheMiddle}(i, j, \mathcal{C}, \mathcal{L})$ which holds if and only if a grid of size $\mathcal{C} \times \mathcal{L}$ contains a unique obstacle (hole or pole) at position (i, j) such that $\upsilon \leq i \leq \mathcal{C}+1-\upsilon$ and $\upsilon \leq j \leq \mathcal{L}+1-\upsilon$ where $\upsilon = (n+1) \times \phi$, namely υ is 3 (resp. 6) for $\mathcal{A}_b^1$ (resp. $\mathcal{A}_b^2$).[3]

III. Extending an Algorithm Adding an Obstacle

We describe our methodology to extend some basis algorithm $\mathcal{A}_b$, originally designed to solve the problem of perpetual exploration of a grid without obstacle with luminous myopic robots that share a common chirality. We assume that a single obstacle is added to the grid, at a node far enough from the grid borders to keep it isolated, see Predicate $\mathit{ObsInTheMiddle}$. The methodology remains the same whatever be the type of the obstacle, pole or hole.

We say that an algorithm $\mathcal{A}_e$ with initial configurations $\mathcal{I}_e$ is an *extension* of Algorithm $\mathcal{A}_b$ with initial configurations $\mathcal{I}_b$ if: (a) $\mathcal{A}_e$ contains at least every rule of $\mathcal{A}_b$, (b) $\mathcal{I}_e$ contains every configuration of $\mathcal{I}_b$ and every configuration of $\mathcal{I}_b$ at which has been added an obstacle at position (i, j) such that $\mathit{ObsInTheMiddle}(i, j)$ holds. Notice that if $\mathcal{I}_b$ is locally-defined, then so is $\mathcal{I}_e$.

Our objective is to synthesize new rules (without modifying existing ones) to guide the robots around the obstacle, so that the whole algorithm, $\mathcal{A}_e$, remains valid and fulfills the perpetual finite grid exploration.

Obviously, the problem being set as above may have far too many solutions to be handled automatically by a tool. For example, there exist 490 rules that can be used to extend $\mathcal{A}_b^1$, and so 2^{490} (namely about 10^{147}) possible extensions have to be considered at first sight, even if, of course, a lot of them are invalid because of incompatibilities between rules. Therefore, we restrict the search space to a user-defined scenario that helps the robots bypass the hole, see Section III-A.

1) This allows to *automatically generate*, a set of valid algorithms $\mathbb{C}$, which are *candidates* to solve the problem.

However, perpetual exploration cannot be guaranteed by our constructions alone due to the nature of the property. To address this, we proceed in two more steps, see Section III-B.

2) We *test by simulation* whether the candidate algorithms achieve perpetual exploration on small grids. To reduce the execution time required for this step, we can restrict the number of simulations to a subset of the synthesized candidates: we select them using the energy consumption of each candidates. Those tests provide the set of *validated* algorithms $\mathbb{A}$.
3) We then extend those results with a *pen-and-paper proof*, showing that every validated algorithm satisfies the perpetual exploration specification for all grid sizes greater than or equal to the tested ones.

We perform a final step to qualitatively analyze the results of the generated algorithms, see Section III-C.

[2] A dead-end obstacle is next to a first wall and at distance two from a second wall.

[3] We may omit some parameters of the predicate if it is clear from the context.

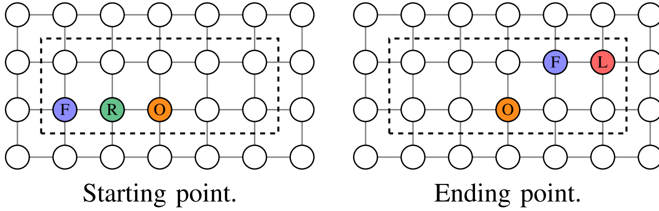

Fig. 3: Goal example for $\mathcal{A}_b^1$.

4) The validated algorithms are passed through a *classification* mechanism to automatically sort them into equivalence classes.

A. Rules Synthesis.

A naive, brute force, way to synthesize rules for new algorithms would be to consider all local views obstacle and generate every possible rule from each of them. Note that considering local views containing the obstacle is not enough since new rules can reach local views without the obstacle that were not considered in the basis algorithm. The rules are then combined together to obtain valid algorithms. Of course this approach would explode since too many possibilities may exist. To make the approach more tractable, we propose to guide the synthesis of the rules in the following way. The designer has to provide a *scenario* that describes how to bypass the obstacle. The idea is that, that when no obstacle in sight, the robots behave as in the basis algorithm; they follow the scenario to react and safely bypass the obstacle; then they resume executing the original algorithm.

To describe a scenario, we first define a list of *goals* where each goal is a pair of local configurations (S, E); S (resp. E) is called the starting (resp. ending) point of the goal. Starting from S, the robots must reach E using existing, but also new synthesized rules. But again, the synthesis of those rules may bring far too many solutions to be handled automatically by a tool; therefore, we bound the problem to make it tractable. For each goal (S, E), we fix the following parameters:

- a rectangular *bounding box* which contains S and E,
- a given number of rounds k allowed to reach E from S.

The idea for the bounding box and k is that, using potentially new synthesized rules, the robots have to go from S to E in at most k rounds; doing so, they have to remain in the nodes defined by the bounding box. For extra control on the trajectory of the robots, some *mandatory nodes* (that at least one robot must visit while going from S to E) or *forbidden nodes* (that no robot should cross while going from S to E) can optionally be defined.

All those parameters (starting points, ending points, bounding boxes, number of rounds, mandatory/forbidden nodes) are *chosen* by the designer based on anticipated trajectories of the robots and are actually called the *scenario*. As an example, Fig. 3 illustrates a goal for $\mathcal{A}_b^1$ where the obstacle (be it a hole or a pole) is shown in orange (O) and the dashed rectangle represents the bounding box.

Once the *scenario* is defined, we automatically generate the rules to obtain all *valid* algorithms matching those constraints. This provides a set of sets of rules on which a filter mechanism is applied and ensures that no two sets of rules are the same or identical up to a permutation of colors. Each remaining set of rules is then used to define a new algorithm, $\mathcal{A}_e$, extension of $\mathcal{A}_b$, the initial configurations $\mathcal{I}_e$ of $\mathcal{A}_e$ being defined as above (see the definition of an *extension*). The set of candidate algorithms $\mathbb{C}$ is exactly made of those extensions $\mathcal{A}_e$.

Note that the list of goals has to be carefully designed. Some situations can be incompatible, *e.g.*, if the scenario contains too many different goals or even if two goals require opposite behaviors in the same local configuration. In those cases, the synthesis tool may fail. On the other hand, the scenario has to cover every possible situation, so that the synthesized algorithms do not block on unexpected local configurations. Overall, a complete scenario is the result of a precise design which can be achieved through an iterative trial-and-error process; this process may be assisted by a tool.

B. Simulations and Proof

After generating the set $\mathbb{C}$ of candidate algorithms through rule synthesis, we know that the algorithms in $\mathbb{C}$ are valid by construction but still have no guarantee that they satisfy the perpetual exploration specification. The final stage of the method aims at building a set $\mathbb{A} \subseteq \mathbb{C}$ of algorithms that solve the problem. But obviously this cannot be fully automatically computed, therefore we proceed in two steps. First we build $\mathbb{A}$ by testing the candidates on a set of grids of small sizes. Second, we extend the guarantee and obtain that every element in $\mathbb{A}$ satisfies the problem.

a) Step 1: Simulations: For each algorithm in $\mathbb{C}$, we run simulations. A simulation requires a given grid, its size and obstacle position being fixed, and an initial configuration. The way we define the set of grids and initial configurations to be tested highly depends on Step 2. During each simulation, the perpetual exploration is tested: if a simulation fails to validate the property, then the algorithm under test is rejected. On the contrary, an algorithm in $\mathbb{C}$ is kept in $\mathbb{A}$ when every simulation (on the whole set of grids and initial configurations) succeeds.

This step can be computationally intensive, when the number of candidate algorithms and the number of grids are big. Therefore, the simulation is run by sets of candidate algorithms $\mathbb{C}^i$ where i is the *level* of the candidate algorithms in $\mathbb{C}^i$. This level is defined as follows. For a candidate algorithm c and a goal g, we use a metric function $m(c, g)$ that computes the *energy* consumed by c to fulfill the goal g, that is, one energy point per movement and color change.[4] The level of c is obtained by computing the sum of the values $m(c, g)$ of all the goals g. The candidates with lower level are simulated first, then the ones with the next level, and so on. The user can choose to run simulations for all levels, or only the first ones.

[4]Our tool can be parameterized to use other metric functions.

b) Step 2: Proof: An algorithm $\mathcal{A}_e \in \mathbb{A}$ was tested in Step 1 on given sets of grids $\mathbf{G}^{\mathtt{sim}}$ and initial configurations $\mathcal{I}^{\mathtt{sim}}$. We extend the simulation results in two steps. We first extend the result to every initial configuration:

Proof Obligation 1: for every $\mathcal{A}_e \in \mathbb{A}$, and every $\mathcal{G} \in \mathbf{G}^{\mathtt{sim}}$, $\mathcal{A}_e$ satisfies the perpetual exploration in $\mathcal{G}$ from every initial configuration in $\mathcal{I}_e$.

This result can be obtained by a pen-and-paper proof which is *ad-hoc* to the current use-case and the condition of the proof provides the definition of $\mathcal{I}^{\mathtt{sim}}$. We define the final result, extending the sizes of the grids in $\mathbf{G}^{\mathtt{sim}}$:

Proof Obligation 2: for every $\mathcal{A}_e \in \mathbb{A}$, and every $\mathcal{G}$ larger than a grid in $\mathbf{G}^{\mathtt{sim}}$, $\mathcal{A}_e$ satisfies the perpetual exploration in $\mathcal{G}$, from every initial configuration in $\mathcal{I}_e$.

Again, this result can be obtained with an *ad-hoc* pen-and-paper proof: we advise to perform this by induction on the size of the grids so that the base case of the induction is provided by Proof Obligation 1. Thus, we can define $\mathbf{G}^{\mathtt{sim}}$ from the requirements of the proof.

C. Classification

To help the user understand what has been generated, a *classification mechanism* is proposed. It gives insights on the generated algorithms and classify them into equivalence classes. Data required to do this classification has been pre-computed during the previous phases: some during the rules synthesis part (*e.g.*, the number of rules in the algorithm), some during the simulation part (*e.g.*, the path used during exploration). Comparisons are made between algorithms according to this pre-computed data, allowing to sort them into equivalence classes.

The first two definitions are the most basic way to determine whether two algorithms are similar: Two algorithms $\mathcal{A}$ and $\mathcal{A}'$ are equivalent if one of the following criteria is met.

- *Identic rules:* they have the same set of rules.
- *Color permutation:* their sets of rules are equal upto a permutation of the colors.

Nonetheless, by construction, our tool generates no pair of extensions that are equivalent under either definition. Thus, we define other criteria to obtain a more detailed comparison of the algorithms. For $i \in \{1, ..., 4\}$, we say that two algorithms $\mathcal{A}$ and $\mathcal{A}'$ are equivalent w.r.t. $\mathcal{C}_i$ iff $\mathcal{C}_i$ actually holds, where

- $\mathcal{C}_1$ – *Number of rules:* $\mathcal{A}$ and $\mathcal{A}'$ have the same number of rules.
- $\mathcal{C}_2$ – *Number of rounds in a cycle:* for every simulated grid and initial configuration, the number of rounds we obtain to perform a cycle of exploration (*i.e.*, coming back to a previously encountered configuration after exploring the whole grid) is the same for both algorithms.
- $\mathcal{C}_3$ – *Energy in a cycle:* for every simulated grid and initial configuration, the energy (number of movements and color changes) spent to perform a cycle of exploration is the same for both algorithms.
- $\mathcal{C}_4$ – *Path in a cycle:* for every simulated grid and initial configuration, at any moment of the cycle of exploration, the robots occupy the same positions in the grid for both algorithms.

Criterion $\mathcal{C}_1$ is purely syntactic. While it provides no piece of information about the behavior of the algorithms in the same class of equivalence, it is obvious that two algorithms that are in different classes according to $\mathcal{C}_1$ are different.

The three other criteria give more insights about the behavior of the algorithms. Notice that $\mathcal{C}_4$ is a refinement of $\mathcal{C}_2$. Indeed, if two algorithms are in the same class under $\mathcal{C}_4$, they use the same path and thus the same number of rounds to perform a cycle: they are in the same class under $\mathcal{C}_2$. On the contrary, $\mathcal{C}_3$ cannot be compared to $\mathcal{C}_2$ nor $\mathcal{C}_4$ due to color changes.

IV. Tool and Case-studies

To demonstrate our methodology, we consider extensions of the two algorithms $\mathcal{A}_b^1$ and $\mathcal{A}_b^2$ to grids with a single obstacle, that is a pole, which has been set far enough away from the walls to avoid impossibility results, *i.e.*, where the predicate $ObsInTheMiddle$ holds. We have developed a tool called **RoASt** (Robot Algorithm Synthesizer) developed in Rust 1.83.0 to support our method. Its code is available at [11]. This tool takes as inputs: a set of rules for an algorithm $\mathcal{A}_b$ that achieves a perpetual exploration of the grid and a scenario that is represented by a set of goals, their bounding boxes, and number of rounds, to guide the tool. It works in 2 phases: the first phase synthesizes all the candidate extension algorithms that satisfy the scenario, see Section III-A and set $\mathbb{C}$; the second phase validates by simulation that the synthesized algorithms are doing a perpetual exploration of some grids from some initial configurations, see Section III-B and set $\mathbb{A}$.

Our aim is to obtain extensions of $\mathcal{A}_b^1$ and $\mathcal{A}_b^2$ that preserve their properties. We consider robots that share a common chirality, with three (resp. two) colors and a visibility range of one (resp. two) for $\mathcal{A}_b^1$ (resp. $\mathcal{A}_b^2$).

Note that this problem is not solvable with only one robot nor with two robots and one color, whatever the visibility. Moreover, it is not solvable with two robots, two colors, and visibility range one. These impossibility results were first demonstrated in [9] with no obstacle; We extended them adding an obstacle, see our technical report [11], Sections II.B and II.C for the proof. Thus, we keep the same settings as in the original algorithms $\mathcal{A}_b^1$ and $\mathcal{A}_b^2$ since the number of robots nor colors can be reduced.

Note also that since $\mathcal{A}_b^1$ and $\mathcal{A}_b^2$ have both locally defined initial configurations, resp. $\mathcal{I}_b^1$ and $\mathcal{I}_b^2$, the sets $\mathcal{I}_e^1$ and $\mathcal{I}_e^2$ of initial configurations of their extensions are both also locally defined, by definition, see the definition of $\mathcal{I}_b$.

In the following, we explain the method in details for $\mathcal{A}_b^1$ and then for each part, provide a recap of the results for $\mathcal{A}_b^2$.

A. Definition of the Goals

The following scenario has been considered for $\mathcal{A}_b^1$. When the robots brush past the obstacle, above or below it, they continue on their way (1). When they arrive in front of the obstacle, their behavior differs according to their color: if the

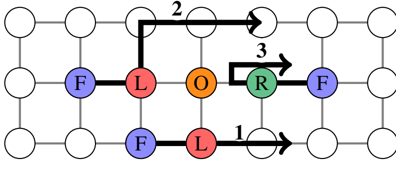

Fig. 4: Considered scenario for $\mathcal{A}_b^1$.

Goal i	$\lvert\mathbb{R}_i\rvert$	Remaining Algorithms after Combination
1	2	
2	2	4
3	2	5
4	2	7
5	7	7
6	18	5
7	1	5
8	1	5

Generation Time: 60 ms – Combination Time: 7 ms

TABLE I: Sets of rules generated for each goal for $\mathcal{A}_b^1$.

Goal i	$\lvert\mathbb{R}_i\rvert$	Remaining Algorithms after Combination
1	1	
2	1	1
3	30	30
4	30	411
5	2	822
6	2	1 644
7	24	39 456
8	24	532 656
9	1	62 208
10	10	373 248
11	2	373 248

Generation Time: 0.3 s – Combination Time: 50 s

TABLE II: Sets of rules generated for each goal for $\mathcal{A}_b^2$.

leader is at color L, they pass over the obstacle by going to the next row (2); otherwise, they turn around and the leader switches color to L (3), see Fig. 4. Overall, this scenario for $\mathcal{A}_b^1$ is made of eight goals.

We define the smallest bounding boxes and number of rounds for each goal that allow to build a solution in the following way. For each goal, we first consider a large number of rounds k and reduce the bounding box as much as possible until it is no more possible to fulfill the goal. Then, with the bounding box being fixed, we reduce k as much as possible until, again, there is no more solution. Note that the three types of parameters are heavily linked. Choosing a certain goal, restricting a bounding box or a number of rounds may invalidate another goal for which no solution is allowed anymore. Defining the whole set of parameters was performed via an iterative trial-and-error process, assisted by the tool.

We used a very similar method for $\mathcal{A}_b^2$ and obtained a scenario of 11 goals. A full description of the goals of each scenario, for $\mathcal{A}_b^1$ and $\mathcal{A}_b^2$, together with their bounded boxes and numbers of rounds is shown in [11].

B. Rules Synthesis

The rules synthesis is done in two phases. First, for optimization purpose, **RoASt** executes a pre-computation phase. Then, it uses the pre-computation results to generate sets of rules that achieves the goals and combine them into valid algorithms.

During the pre-computation phase, **RoASt** first generates all possible (local) views without walls, eliminating indistinguishable views and those already covered by the rules of $\mathcal{A}_b$ (this prevents nondeterminism). We obtain 35 views for $\mathcal{A}_b^1$. Then, **RoASt** computes all possible rules, *i.e.*, for each of the 35 views, it associates every possible movement (including remaining idle) and every possible color change (including keeping the same color). We obtain 490 rules. Finally, **RoASt** computes all possible *parallel rules*: a parallel rule is composed of a local configuration and, for each robot in this configuration, whether the robot moves and/or changes its color, and if so, its direction and new color. It results in 2449 parallel rules.

Then, these pre-computed parallel rules are used by **RoASt** to generate, for each goal i, the set $\mathbb{R}_i$ of all possible sets of rules that can achieve the goal. It ensures that the swarm can reach the ending point of the goal from the starting point in less than k rounds, remains within the bounding box, and respect the mandatory/forbidden nodes. Table I shows the size of each set $\mathbb{R}_i$ for $\mathcal{A}_b^1$.

After all the sets $\mathbb{R}_i$ are generated, they are iteratively combined to each other: $\mathbb{R}_1$ is combined with $\mathbb{R}_2$, then the result of this combination is itself combined with $\mathbb{R}_3$, and so on. Some combinations are not valid, *e.g.*, the rules used are incompatible, and are eliminated ensuring that the generated algorithms are valid by construction. For example, when combining $\mathbb{R}_6$ of $\mathcal{A}_b^1$ with the sets of rules resulting from the combination of the five first sets $\mathbb{R}_i$, we do not have $7 \times 18 = 126$ possible algorithms (Cartesian product of the remaining possible algorithms after Goal 5 and $\mathbb{R}_6$), but only five valid algorithms as shown on the third column of Table I.

Among the $2016 = 2 \times 2 \times 2 \times 2 \times 2 \times 7 \times 18 \times 1 \times 1$ possible sets of rules for $\mathcal{A}_b^1$ (the Cartesian product of the possibilities for each goal), only five valid algorithms remain in $\mathbb{C}_1$ at the end of the synthesis.

On an AMD EPYC 7763 64-core @ 256 $\times$ 2.45GHz (2TB of RAM) running Debian 12.11 with Linux 6.1.0-37-amd64, the pre-computation for $\mathcal{A}_b^1$ lasts 611 ms. Then, the generation of all the sets $\mathbb{R}_i$ lasts 60 ms and their combination lasts 7 ms. Thus, the five valid algorithms of $\mathbb{C}_1$ are generated in 2 seconds.

The exact same process is used to compute the set $\mathbb{C}_2$ of candidate extensions for $\mathcal{A}_b^2$. But the larger visibility range leads to a more intensive computation to explore the larger number of possible sets of rules. Indeed, **RoASt** generates 146 views, 1298 rules, and 5892 parallel rules during the pre-computation phase. Table II shows the sets $\mathbb{R}_i$ generated for each of the 11 goals and the result of their combination. We obtain 373 248 candidate algorithms in $\mathbb{C}_2$. Using the same computer as previously, this generation lasts approximately 1 minute: 5 s for the pre-computation, 0.3 s for the generation of sets $\mathbb{R}_i$, and 50 s for their combination.

Notice that the order of the goals does not change the

Goal i	$\lvert\mathbb{R}_i\rvert$	Remaining Algorithms after Combination
3	30	
4	30	411
7	24	9 864
8	24	133 164
10	10	839 268
5	2	1 645 272
6	2	3 257 280
11	2	3 257 280
1	1	3 257 280
2	1	3 257 280
9	1	373 248

Generation Time: 0.4 s – Combination Time: 5 min

TABLE III: Sets of rules generated for each goal for $\mathcal{A}_b^2$, using a different order of the goals.

generated algorithms, but it may impact the duration of the combination phase because of the number of the remaining sets of rules at the end of each combination. See for example Table III that shows the combination of sets $\mathbb{R}_i$ for $\mathcal{A}_b^2$ with the same goals than the one used in Table II, but in a different order. While the pre-computation and generation of sets $\mathbb{R}_i$ lasts the same time, the combination is five time longer, because it manipulates a larger number of remaining algorithms during the intermediate steps (more than 3 millions remaining algorithms at most, instead of only about 5 hundred thousands with the previous goal order). Those results are available at [11].

C. Validation by Simulation

The candidate algorithms in $\mathbb{C}_1$ are validated on small size grids by simulation. We define the grids to be used by the simulation tests as the set $\mathbf{G}_1^{\mathtt{sim}}$ which contains the 27 grids of size 7×7,[5] 7×8, 7×9, 8×7, 8×8, 8×9, 9×7, and 9×8 with the obstacle at position (i, j) such that $ObsInTheMiddle(i, j, \mathcal{C}, \mathcal{L})$.

We also define the initial configurations to be used by the simulation tests, noted $\mathcal{I}_1^{\mathtt{sim}}$, as the ones where the two robots are side-by-side, one with color F, the other with color L or R and one of them has the obstacle in sight. Overall, $\mathcal{I}_1^{\mathtt{sim}}$ contains 12 initial configurations.

For each grid $\mathcal{G} \in \mathbf{G}_1^{\mathtt{sim}}$, each initial configuration $\gamma \in \mathcal{I}_1^{\mathtt{sim}}$, and each algorithm $\mathcal{A} \in \mathbb{C}_1$, $\mathcal{A}$ is simulated on $\mathcal{G}$ starting from γ during $\mathtt{N}$ rounds (the value of $\mathtt{N}$ is discussed later). If the robots stop their execution before the end of the $\mathtt{N}$ rounds, $\mathcal{A}$ is rejected. Otherwise, we check if there is a cycle in the execution, *i.e.*, if the last configuration appears at least twice in the execution. If a cycle is detected, there are two cases.

(a) If the grid was not completely explored during the cycle, $\mathcal{A}$ is rejected.
(b) If the whole grid was explored during the cycle, $\mathcal{A}$ is validated.

If no cycle is detected, the simulation is extended by $\mathtt{N}$ additional rounds until one of the three previous cases is reached. We arbitrarily chose to fix $\mathtt{N}$ at three times the size of the larger grid of $\mathbf{G}_1^{\mathtt{sim}}$, so $\mathtt{N} = 3 \times 9 \times 8 = 216$ rounds.

[5] 7×7 is the smallest grid on which we can set an obstacle such that $ObsInTheMiddle$ holds.

Level i	Energy	$\lvert\mathbb{C}^i\rvert$	Validated Algos	Validation Duration
1	37	1	1	181 ms
2	41	2	2	286 ms
3	44	1	1	184 ms
4	46	1	1	183 ms
5		0	0	

Total Validation for all levels Time: 0.8 s

TABLE IV: Result of the validation phase for $\mathcal{A}_b^1$.

Level i	Energy	$\lvert\mathbb{C}^i\rvert$	Validated Algos	Validation Duration
1	75	7	5	5 s
2	77	159	24	1 min
3	79	1529	44	9 min
4	81	8216	42	52 min

Validation Time until Level 4: 1 h 2 min

TABLE V: Result of the validation phase for $\mathcal{A}_b^2$.

As explained in Section III-B, the validation is done by sets of candidate algorithms according to their level of energy used to fulfill the goals. For $\mathcal{A}_b^1$, the five candidate algorithms are distributed into four levels with energy 37, 41, 44 and 46 respectively. For each candidate, in each level, $|\mathbf{G}_1^{\mathtt{sim}}| \times |\mathcal{I}_1^{\mathtt{sim}}| = 27 \times 12 = 324$ simulations were required. Table IV shows the duration of this validation and the number of validated algorithms for each level. We can see that all five candidate algorithms passes the validation phases in 0.8 s. The rules of the five validated algorithms are given in our technical report [11], Section III and their animation are also available on [11]. Therefore, we have the following lemma, where $\mathbb{A}_1$ is composed of these five validated algorithms.

Lemma 1. *Let $\mathcal{A} \in \mathbb{A}_1$, $\mathcal{G} \in \mathbf{G}_1^{\mathtt{sim}}$, $\gamma \in \mathcal{I}_1^{\mathtt{sim}}$. The execution of $\mathcal{A}$ from γ satisfies the perpetual exploration of $\mathcal{G}$.*

Again, the same validation process is used for the candidate algorithms in $\mathbb{C}_2$ for $\mathcal{A}_b^2$. The grids used by the simulation (set $\mathbf{G}_2^{\mathtt{sim}}$) contains the 27 grids of size 13×13, 13×14, 13×15, 14×13, 14×14, 14×15, 15×13, and 15×14 with the obstacle at position (i, j) such that $ObsInTheMiddle(i, j, \mathcal{C}, \mathcal{L})$. The set of initial configurations used for the simulations, $\mathcal{I}_2^{\mathtt{sim}}$, is defined as the ones where the two robots are colinear, at distance one or two, one with color L, the other with color F, and at least one of them has the obstacle in sight; it contains 34 initial configurations.

The number of candidate algorithms is far greater for $\mathcal{A}_b^2$ than for $\mathcal{A}_b^1$, with 373 248 candidate distributed into 12 levels of energy. We chose to run the validation only on the first four levels, just as for $\mathcal{A}_b^1$. For each candidate in these levels, $27 \times 34 = 918$ simulations were required. Table V shows the results of these simulations. After approximately one hour, it results in a set $\mathbb{A}_2$ of 115 validated algorithms. Their animations are available at [11]. We have the following lemma.

Lemma 2. *Let $\mathcal{A} \in \mathbb{A}_2$, $\mathcal{G} \in \mathbf{G}_2^{\mathtt{sim}}$, $\gamma \in \mathcal{I}_2^{\mathtt{sim}}$. The execution of $\mathcal{A}$ from γ satisfies the perpetual exploration of $\mathcal{G}$.*

D. Proof of Perpetual Exploration

Let $i \in \{1, 2\}$. We denote by $size_i$ the smallest value of a grid dimension in which extensions of $\mathcal{A}_i$ perform the

Equivalence Definition	Number of Classes	Number of Algorithms by Class
$\mathcal{C}_1$ – Number of rules	3	2 classes of 2, 1 class of 1
$\mathcal{C}_2$ – Number of rounds in a cycle	1	1 class of 5
$\mathcal{C}_3$ – Energy in a cycle	5	5 classes of 1
$\mathcal{C}_4$ – Paths in a cycle	3	2 classes of 2, 1 class of 1

TABLE VI: Classification of $\mathbb{A}_1$ (5 algorithms).

Equivalence Definition	Number of Classes	Number of Algorithms by Class
$\mathcal{C}_1$ – Number of rules	2	1 class of 61, 1 class of 54
$\mathcal{C}_2$ – Number of rounds in a cycle	2	1 class of 32, 1 class of 83
$\mathcal{C}_3$ – Energy in a cycle	6	1 class of 11, 1 class of 6 1 class of 48, 1 class of 8 1 class of 24, 1 class of 18
$\mathcal{C}_4$ – Paths in a cycle	39	13 classes of 4, 7 classes of 1 16 classes of 2, 3 classes of 8

TABLE VII: Classification of $\mathbb{A}_2$ (115 algorithms).

perpetual exploration. For algorithms of $\mathbb{A}_1$, the smallest grid we consider is 7×7, thus $size_1 = 7$. Similarly, $size_2 = 13$.

To prove that every valid generated extensions perform the perpetual exploration, we need to prove the following results. For lack of space, we only give here a sketch of the proofs. The complete proof is in our technical report [11], Section I.

We first fulfill Proof Obligation 1 and extend the simulation results to any (locally-defined) initial configuration, $\mathcal{I}_e^i$.

Lemma 3. *Let $i \in \{1, 2\}$. Let $\mathcal{A} \in \mathbb{A}_i$, $\mathcal{G} \in \mathbf{G}_i^{\mathtt{sim}}$, $\gamma \in \mathcal{I}_e^i$. The execution of $\mathcal{A}$ starting from γ satisfies the perpetual exploration of $\mathcal{G}$.*

Sketch of proof. The idea of the proof is that, if $\gamma \notin \mathcal{I}_i^{\mathtt{sim}}$, the robots eventually reach some configuration of $\mathcal{I}_i^{\mathtt{sim}}$, from which they perform the perpetual exploration using the results of Lemmas 1 and 2.

Then, these results are extended to larger grids, proving Proof Obligation 2.

Theorem 2. *Let $i \in \{1, 2\}$. Let $\mathcal{A} \in \mathbb{A}_i$. For every grid $\mathcal{G}$ of size $\mathcal{C} \times \mathcal{L}$, where $\mathcal{C}, \mathcal{L} \geq size_i$, with an obstacle in node (x, y) such that $ObsInTheMiddle(x, y, \mathcal{C}, \mathcal{L})$, for every configuration $\gamma \in \mathcal{I}_e^i$, the execution of $\mathcal{A}$ starting from γ satisfies the perpetual exploration of $\mathcal{G}$.*

Sketch of proof. The proof is done by induction on the size of the grid. The base cases are covered by Lemma 3 (grids of $\mathbf{G}_i^{\mathtt{sim}}$). Then, the induction step consists in adding two extra columns or rows and showing that perpetual exploration is also performed in the larger grid. This is done by exploiting the properties of the basis algorithm $\mathcal{A}_i$: when far enough away from the walls and the obstacle, the robots cannot distinguish if they are in the smaller or the larger grid and repeat the same pattern of movement. Thus if they perform perpetual exploration in the smaller grid, they also do so in the larger grid. Notice that columns and rows are added by two (and not only one) since the behavior of the robots depends on the parity of the grid dimensions.

E. Analysis of the results

To obtain insights about the results of the synthesis, the five valid extensions for $\mathcal{A}_b^1$ and 115 valid extensions for $\mathcal{A}_b^2$ were run through the classifier, leading to the results presented in Tables VI and VII, respectively. The detailed results are shown at [11]. This classification shows that, while every generated extension is different from one another, some of them have similarities. Let us focus on the results for $\mathcal{A}_b^2$.

First, notice that there is only two classes of algorithms under criterion $\mathcal{C}_1$, the number of rules. In the first class, all the 54 algorithms are composed of 151 rules while the 61 algorithms of the second class are composed of 152 rules. This gives no indication on how the algorithms behave, yet it highlights the high combinatorics faced by the synthesis: with such a little difference in their rule numbers, we obtain more than one hundred different valid algorithms.

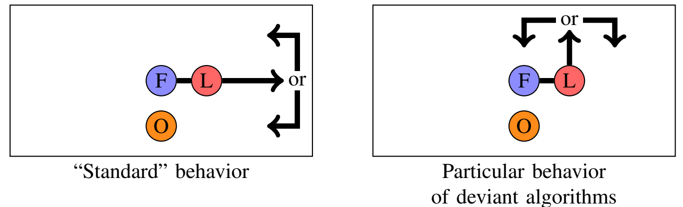

Fig. 5: Major difference in the behaviors of algorithms in $\mathbb{A}_2$.

Now, if we consider the number of rounds in a cycle ($\mathcal{C}_2$), we also have only two classes. This time, the behaviors of the algorithms explain the classification. Indeed, it appears that the 32 algorithms in the first class always have a “standard” behavior when passing next to the obstacle: they pass the obstacle and then continue on the same *e.g.* row. The other algorithms have the same behavior most of the time, except in some particular grids, and starting from particular initial configurations. From those configurations, the robots change the previous direction of exploration, *e.g.*, from exploring the grid horizontally, row by row, they explore it vertically, column by column (see Fig. 5). In the remaining, these algorithms will be referred as *deviant*.[6] This difference can be seen on the simulations of the generated Algorithms 3 and 5 on Experiment 3 [11], for example.

This difference of behavior also impacts the classification according to the energy consumption in a cycle ($\mathcal{C}_3$). However, algorithms that are in the same class according to $\mathcal{C}_2$ are split into three different classes each, according to $\mathcal{C}_3$, leading to a total of six classes. We denote by $\mathcal{C}_3^{d,i}$ (resp. $\mathcal{C}_3^{n,i}$) , $i \in \{1, 2, 3\}$ the three classes of deviant (resp. non-deviant) algorithms. We can observe that each set of those three classes are ordered $\mathcal{C}_3^{d,1} \leq \mathcal{C}_3^{d,2} \leq \mathcal{C}_3^{d,3}$ and $\mathcal{C}_3^{n,1} \leq \mathcal{C}_3^{n,2} \leq \mathcal{C}_3^{n,3}$, meaning that the algorithms of $\mathcal{C}_3^{d,1}$ are always[7] more efficient than the algorithms of $\mathcal{C}_3^{d,2}$, and so on. The difference on

[6] Remark that the partitions into classes under criteria $\mathcal{C}_1$ and $\mathcal{C}_2$ are completely independent. The number of rules does not indicate whether the algorithm is deviant or not, and conversely.

[7] The comparison is made on every simulation, *i.e.* for every grid and initial configuration used in the previous step of our method.

Goal i	$\|\mathbb{R}_i\|$	Remaining Algorithms after Combination
1	2	
2	2	4
3	2	5
4	2	7
5	7	7
6	18	5
7	1	5
8	4	15

Generation Time: 64 ms – Combination Time: 7 ms

TABLE VIII: Sets of rules generated for each goal for $\mathcal{A}_b^1$, with the same goals than for Table I, but allowing one extra round for Goal 8.

Level i	Energy	$\|\mathbb{C}^i\|$	Validated Algos	Validation Duration
1	37	1	1	135 ms
2	39	2	2	222 ms
3	41	2	2	222 ms
4	43	4	4	422 ms
5	44	1	1	169 ms
6	46	3	3	401 ms
7	48	2	2	283 ms
8		0		

Total Validation Time: 2.892 s

TABLE IX: Result of the validation phase for $\mathcal{A}_b^1$, with the same goals than for Table IV, but allowing one extra round for Goal 8.

the energy consumption of two deviant or two non-deviant algorithms is due to the fact that some algorithms involve additional change of colors or robots following a longer path. To observe the difference, the reader can consult for example the simulation of Algorithms 3, 31, and 576 (respectively in classes $\mathcal{C}_3^{d,1}$, $\mathcal{C}_3^{d,2}$, and $\mathcal{C}_3^{d,3}$) on Experiment 4, see [11].

Similarly, if we consider Definition $\mathcal{C}_4$, the deviant algorithms are split into 24 classes and the non-deviant algorithms are split into 15 other classes. This is due to little differences in the followed paths. For example, the reader can consult the simulation of Algorithms 3 and 26 on Experiment 0, see [11]. On this example, the leader first moves away from the follower before the robots move along keeping a distance two between them in Algorithm 26, yet they stay side-by-side at distance one in Algorithm 3.

Notice that there is no correspondence between the classes according to $\mathcal{C}_3$ and $\mathcal{C}_4$. Indeed, two algorithms can use the same paths, but consume a different amount of energy if one of them requires more color changes.

F. Considering Other Parameters

In this part we show that our method and tool are easily able to generate other sets of algorithms, by changing the setting of parameters.

a) Extending the number of rounds in a goal: In the case-study described before, we considered goals with small bounding boxes and numbers of rounds (namely the smallest we were able to design and obtain results). It gave us five algorithms for $\mathcal{A}_b^1$. But by changing those parameters, we can generate other algorithms. For example, if we keep the same goals, but only increase the number of rounds allowed on the last goal (Goal 8, where the robots are on the same row than the obstacle and move away from it), we have four possibilities to fulfill that goal (instead of only one previously) leading to 15 candidate algorithms: the five previous algorithms plus 10 additional ones, see Table VIII. Just as in Section IV-C, the candidate algorithms are validated on small grids by simulation and each of them passes this validation phase (lasting 2.892 sec) leading to a set $\mathbb{A}_1'$ of 15 validated algorithms, see Table IX. Notice that there are three extra levels of energy. The new goals and the results are available at [11]. Notice that the proofs of Section IV-D also hold for these algorithms.

b) Dealing with a hole instead of a pole: All along the case study we focus on the obstacle being a *pole*. We here present an experiment with a *hole*. This means that the robots still have to bypass the obstacle but, while a pole hides what is behind it, a robot can observe the grid beyond a hole, when its visibility enables it. The tool **RoASt** has an option to choose whether the obstacle is a hole or a pole.

When the visibility range is limited to one, it makes no difference whether the robot sees a hole or a pole. Thus, we can relaunch the same experiments than previously on $\mathcal{A}_b^1$ using the same scenarios as in IV-A and observe the results, see [11]. As expected, the algorithms obtained are exactly the same. On the contrary, when the visibility range is two, things are different. Since the robots can see through the hole, an additional initial configuration is required for $\mathcal{A}_b^2$, namely the one where the robots are on either side of the hole. Yet, we still obtain 115 extension algorithms: they are the same 115 algorithms as before where for each one some rules have been added to handle the situation where the robots see each other through the hole. Surprisingly maybe, this situation added no more combinatorial choices that may have created other algorithms. Those algorithms are available at [11].

c) Other scenarios of $\mathcal{A}_b^2$: Again, using the tool parameters settings, we experimented on $\mathcal{A}_b^2$ with two different other scenarios where the obstacle is still a pole. For both scenarios, we reuse the standard goals as introduced in Section IV-A with one goal slightly modified each time.

Firstly, we modified the target position of Goal 11 by exchanging the colors of the two robots (from F to L and conversely). Previously, this goal enforced the robots to move up and turn in the opposite direction when facing the obstacle. This change allows the robots to continue in the same direction than the one they were facing when coming to the obstacle. We obtained 85 algorithms in 23 min (against 1h 5 min for the standard experiment), see [11]. By allowing the robots to continue forward when they bypass the obstacle, they get into similar situations that ones faced in other goals, such as Goal 7 where they brush past the obstacle. This leads to more incompatibilities when combining the set of rules generated for these goals, and thus, to less algorithms.

Secondly, we modified the target position of Goal 9: we moved the robots two positions away from the obstacle and their initial position. As a consequence we also enlarged the bounding box by one and the allowed number of rounds by two. (Goal 9 is the one where the robots and the obstacle

are aligned, with the leader robot at a distance of two from the obstacle.) We obtained 475 algorithms, see [11]. Since we enlarged the bounding box, it gave more freedom for the robots to move. As a result we obtained more algorithms but at the cost of a significantly increased execution time (about three hours and 7 minutes). Nonetheless, it leads to original behaviors that may not have been used in a by-hand designed algorithms. For example, the reader can consult the animation of Algorithm 188 on Experiment 26 [11]. In this example, when facing the obstacle, the leader moves up while the follower (that was at distance two) continue in direction of the obstacle. The robots are thus on a diagonal and moves towards the obstacle remaining so, until the follower is next to the obstacle and moves up.

Notice that if we combine both changes, the one on Goal 11 and the one on Goal 9, due to incompatibilities between these two goals, we obtain only 181 algorithms, see [11].

V. Conclusion and perspectives

The method we proposed has proven itself quite efficient in our case-study, needing only a few seconds on a standard computer to synthesize fifteen new algorithms optimal in number of colors and robots under visibility range one; a tremendous speed up compared to a naive approach that would have to explore approximately 10^{147} possible solutions. Visibility range two is more computationally intensive, but our heuristics allows for example to synthesize 115 new algorithms in a little more than one hour. These results are quite promising for future works tackling more difficult situations, *e.g.*, larger visibility range, obstacles spanning several nodes of the grid, or triangular grids.

Nonetheless, this approach does not generate all possible algorithms since it relies on the design of a scenario to bypass the hole. The work of the designer remains important and greatly impacts the result. Even with the guidelines provided in our methodology, some set of goals may generate algorithms that do not perform a perpetual exploration and thus fail at the validation by simulation step. For example, let consider what happens if we change only Goal 6 for $\mathcal{A}_b^1$ (when the robots face the obstacle and bypass it moving up with the leader of color L) enforcing the leader to switch to color R. It appears that we also generate five candidate algorithms, but none of them pass the simulation phase. The reader can consult the animations of those candidate algorithms on [11] to see that the robots cycle around the obstacle without exploring the whole grid.

The parameters of each goal (the bounding box and the number of rounds k) also have a huge impact, not only on the resulting algorithms but also on the performances: the larger the bounding box and k are, the more possibilities there are to explore, the larger the generation time. As a future work, we would like to explore the automatic design of scenarios.

Finally, we believe that the method presented in this work opens the way to generating distributed algorithms for swarm of robots *from scratch*, rather than as extensions of an existing algorithm.

References

[1] The Rocq Development Team, “The Rocq reference manual – release 9.0.0,” 2025. [Online]. Available: https://rocq-prover.org/doc/V9.0.0/

[2] The Pactole Project, “Pactole framework.” [Online]. Available: https://pactole.liris.cnrs.fr/

[3] B. Bérard, P. Lafourcade, L. Millet, M. Potop-Butucaru, Y. Thierry-Mieg, and S. Tixeuil, “Formal verification of mobile robot protocols,” *Distributed Comput.*, vol. 29, no. 6, pp. 459–487, 2016.

[4] X. Défago, A. Heriban, S. Tixeuil, and K. Wada, “Using model checking to formally verify rendezvous algorithms for robots with lights in euclidean space,” *Robotics Auton. Syst.*, vol. 163, p. 104378, 2023.

[5] F. Bonnet, X. Défago, F. Petit, M. Potop-Butucaru, and S. Tixeuil, “Discovering and assessing fine-grained metrics in robot networks protocols,” in *SRDS*, 2014, pp. 50–59.

[6] L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil, “On the synthesis of mobile robots algorithms: The case of ring gathering,” in *SSS*, 1980, pp. 237–251.

[7] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder, “Decidability in parameterized verification,” *SIGACT News*, vol. 47, no. 2, pp. 53–64, 2016.

[8] Q. Bramas, S. Devismes, A. Durand, P. Lafourcade, and A. Lamani, “Optimal asynchronous perpetual grid exploration,” in *SSS*, 2024, pp. 89–105.

[9] Q. Bramas, P. Lafourcade, and S. Devismes, “Optimal exclusive perpetual grid exploration by luminous myopic opaque robots with common chirality,” in *ICDCN*, 2021, pp. 76–85.

[10] A. Rauch, Q. Bramas, S. Devismes, P. Lafourcade, and A. Lamani, “Optimal exclusive perpetual grid exploration by luminous myopic robots without common chirality,” in *NETYS*, 2021, pp. 95–110.

[11] K. Altisen, A. Durand, P. Lafourcade, and O. Nahnah, “Code of **RoASt** tool and additional content,” 2026. [Online]. Available: https://luminousrobots.github.io/RoASt-Docs

[12] F. Bonnet, A. Milani, M. Potop-Butucaru, and S. Tixeuil, “Asynchronous exclusive perpetual grid exploration without sense of direction,” in *OPODIS*, 2011, pp. 251–265.

[13] S. Nagahama, F. Ooshita, and M. Inoue, “Terminating grid exploration with myopic luminous robots,” in *IPDPS Workshops*, 2021, pp. 586–595.

[14] G. Prencipe, “Instantaneous actions vs. full asynchronicity : Controlling and coordinating a set of autonomous mobile robots,” in *ICTCS*, vol. 2202, 2001, pp. 154–171.

[15] M. Sardar, D. Das, and S. Mukhopadhyaya, “Exploration of a grid with blocked nodes using a swarm of autonomous robots,” in *PCCDA*, 2024, pp. 15–31.

[16] M. Potop-Butucaru, N. Sznajder, S. Tixeuil, and X. Urbain, “Formal methods for mobile robots,” in *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, 2019, pp. 278–313.

[17] F. Bonnet, Q. Bramas, P. Courtieu, X. Défago, L. Rieg, S. Tixeuil, and X. Urbain, “Deterministic color-optimal self-stabilizing semi-synchronous gathering: A certified algorithm,” in *SIROCCO*, 2025, pp. 127–143.

[18] T. Balabonski, A. Delga, L. Rieg, S. Tixeuil, and X. Urbain, “Synchronous gathering without multiplicity detection: a certified algorithm,” *Theory Comput. Syst.*, vol. 63, no. 2, pp. 200–218, 2019.

[19] T. Balabonski, R. Pelle, L. Rieg, and S. Tixeuil, “A foundational framework for certified impossibility results with mobile robots on graphs,” in *ICDCN*, 2018, pp. 5:1–5:10.

[20] P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain, “Swarms of mobile robots: Towards versatility with safety,” *Leibniz Trans. Embed. Syst.*, vol. 8, no. 2, pp. 02:1–02:36, 2022.

[21] D. Peleg, “Distributed coordination algorithms for mobile robot swarms: New directions and challenges,” in *IWDC*, vol. 3741, 2005, pp. 1–12.

[22] I. Suzuki and M. Yamashita, “Distributed anonymous mobile robots: Formation of geometric patterns,” *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1347–1363, 1999.

[23] Q. Bramas, P. Lafourcade, and S. Devismes, “Animation of the first algorithm in [9],” 2021. [Online]. Available: https://bramas.fr/static/ICDCN2021/2-robots-3-colors-range-1.html

[24] ——, “Animation of the second algorithm in [9],” 2021. [Online]. Available: https://bramas.fr/static/ICDCN2021/2-robots-2-colors-range-2.html