Optimal Torus Exploration by Oblivious Mobile Robots

S. Devismes[1], A. Lamani[2], F. Petit[3], and S. Tixeuil[3]

[1] VERIMAG UMR 5104, Université Joseph Fourier, France
[2] MIS, Université de Picardie Jules Verne (France)
[3] LIP6, UPMC Sorbonne Universités (France)

Abstract

We consider autonomous robots that are endowed with motion actuators and visibility sensors. The robots we consider are weak, *i.e.*, they are anonymous, uniform, unable to explicitly communicate, and oblivious (they do not remember any of their past actions). In this paper, we propose an optimal (*w.r.t.* the number of robots) solution for the terminating exploration of torus-shaped networks by a team of k such robots in the SSYNC model.

In more details, we first show that it is impossible to explore any simple torus of arbitrary size with (strictly) less than four robots, even if the algorithm is probabilistic. If the algorithm is required to be deterministic, four robots are also insufficient. This negative result implies that the only way to obtain an optimal algorithm (*w.r.t.* the number of robots participating to the algorithm) is to make use of probabilities.

Then, we propose a probabilistic algorithm that uses four robots to explore all simple tori of size $\ell \times L$, where $7 \leq \ell \leq L$. Hence, in such tori, four robots are necessary and sufficient to solve the (probabilistic) terminating exploration. As a torus can be seen as a 2-dimensional ring, our result shows, perhaps surprisingly, that increasing the number of possible symmetries in the network (due to increasing dimensions) does not necessarily come at an extra cost *w.r.t.* the number of robots that are necessary to solve the problem.

Keywords: Robot, Torus, Exploration, Obliviousness

1 Introduction

We consider autonomous robots that are endowed with motion actuators and visibility sensors, but that are otherwise unable to communicate. They evolve in a discrete environment, *i.e.*, the space is partitioned into a finite number of locations, conveniently represented by a graph, where the nodes represent the possible locations that a robot can take and the edges the possibility for a robot to move from one location to another.

Those robots must collaborate to solve a collective task despite being limited with respect to inputs from the environment, asymmetry, memory, *etc*. In particular, the robots we consider are anonymous, uniform, yet they can sense their environment and take decisions according to their own ego-centered view. In addition, they are oblivious, *i.e.*, they do not remember their past actions. Robots operate in *cycles* that include three phases: *Look*, *Compute*, and *Move* (L-C-M, for short). The Look phase consists in taking a snapshot of the other robots positions using its visibility sensors. During the Compute phase, a robot computes a target destination based on the previous observation. The Move phase simply consists in moving toward the computed destination using motion actuators. Using L-C-M cycles, three models has been introduced in the literature, capturing the various degrees of synchrony between robots. According to a recent taxonomy [1], they are denoted FSYNC, SSYNC, and ASYNC, from the stronger to the weaker. The former stands for

fully synchronous. In this model, all robots execute the L-C-M cycle synchronously and atomically. In the SSYNC (*semi-synchronous*) model, robots are asynchronously activated to perform cycles, yet at each activation, a robot executes one cycle atomically. With the weaker model, ASYNC (stands for *asynchronous*), robots execute L-C-M in a completely independent manner.

In this context, typical problems are *terminating exploration* [2, 3, 4, 5, 6], *exclusive perpetual exploration* [7, 8, 9], *exclusive searching* [10, 9], and *gathering* [9, 11, 12]. In this paper, we address the *terminating exploration* (or simply *exploration*) problem, which requires that robots collectively explore the whole graph and stop upon completion. We focus on the case where the network is an *anonymous unoriented torus* (or simply *torus*, for short). The terms *anonymous* and *unoriented* mean that no robot has access to any kind external device (*e.g.*, node identifiers, oracle, local edge labeling) allowing to identify nodes or to determine any (global or local) direction, such as North-South/East-West.

A question naturally arises: "*Why addressing an abstract topology such as torus?*" To answer this question, we must emphasize that robots are unable to communicate explicitly and have no persistent memory. So, they are unable to remember the various steps taken before. Therefore, the positions of the other robots are the only way to distinguish the different stages of the exploration process. Torus belongs to the class of *regular* graphs, *i.e.,* graphs where each vertex has the same number of neighbors. Such graphs are of particular interest because they are topologies for which the symmetry of configurations with respect to robot positions is the most frequently observed, making the exploration problem hard to solve. So far, ring-shaped network is the only *regular* topology that has been studied [13, 6, 4]. As a result, an immediate question arises: "*Does the increase of the number of possible symmetries in the network (mainly due to increasing dimensions) make the problem harder to solve?*" Terminating exploration has been studied in other topologies than rings, namely the tree [5] and grid [3]. However, none of them are regular networks. Torus can be seen as a 2-dimensional ring. Compared to the ring, the main difficulty lies in the additional axes of symmetry. It appears to be the most natural candidate among regular graphs to study the impact of strong topological symmetry on the complexity to solve the problem.

Furthermore, as previously stated, the exploration (with stop) process is intrinsically related to the ability to differentiate consecutive phases of the exploration. More possible symmetries hint that more robots than in rings are required to complete exploration: As robots have no way to distinguish and agree on some kind of orientation, *e.g.*, North-South/East-West, somehow the current robot configuration has to encode consistent information so that robots agree on both axes. Since numerous symmetric configurations induce a large number of required robots, minimizing the number of robots turns out to be a difficult problem.

Related Work. With respect to the (terminating) exploration problem, minimizing the number of robots for exploring particular classes of graphs led to contrasted results.

The only result available for exploration in general graphs [2] considers that edges are labeled in such a way that the network configuration (made of the topology, the edge labeling, and the robot positions) is asymmetric. In this extended model, three robots are not sufficient to explore all asymmetric configurations, and four robots are sufficient to explore all asymmetric configurations. Note that exploring the set of asymmetric configurations is strictly weaker than exploring the complete underlying graph, especially when the graph is highly symmetric.

The rest of the literature is thus dedicated to a weaker model, where edges are not labeled (or equivalently, the labeling is decided by an adversary anytime a robot is activated). One extreme case in this weak model is the set of tree-shaped networks, as in general, $\Omega(n)$ robots are necessary and sufficient to explore a tree network of n nodes deterministically [5]. The other extreme case is the set of grid-shaped networks [3], where three robots are necessary and sufficient to explore deterministically any grid of at least three nodes (except for the grids of size 2×2 and 3×3, where four – respectively five – robots are necessary and sufficient). However, this result is mainly due to the fact that grids are not regular graphs: they contain nodes of

degrees 2, 3, and 4. This topological property implies less symmetries.

In contrast, rings and tori are regular graphs, and consequently more intricate. In ring-shaped networks [6], the fact that the number k of robots and the ring size n must be coprime yields to the lower bound $\Omega(\log n)$ on the number of robots required to explore a n-size ring. Indeed, the smallest non-divisor of n evolves as $\log n$ in the worst case. However, notice that Lamani *et al.* also provide in [13] a protocol that allows 5 robots to deterministically explore any ring whose size is coprime with 5. The large number of robots and the constraint on the ratio between the number of robots and the ring size induced by the deterministic setting in ring-shaped networks hinted at a possible more efficient solutions when robots can make use of probabilities [4]. As a matter of fact, four robots are necessary and sufficient to probabilistically explore any ring of size at least four. While the gain in going probabilistic is only one robot when n is not divisible by 5, a logarithmic factor is obtained in the general case. Aforementioned deterministic solutions typically operate in the ASYNC model, while probabilistic ones can only cope with the weaker SSYNC model. Actually, an impossibility result [4] explicitly states that randomization does not help in the ASYNC model (that is, there exists a scheduling such that random choices are all nullified).

So far, no research explored the feasibility of exploring a torus-shaped network with a team of k robots. The exploration of tori is a step forward toward exploration of other (maybe more complex) *unoriented periodic 2D discrete spaces*, *e.g.*, spheres.

Contribution. We propose an optimal (*w.r.t.* the number of robots) solution for the terminating exploration of torus-shaped networks by a team of k such robots. In more details, we first show that it is impossible to explore any simple torus of arbitrary size with less than four robots, even if the algorithm is probabilistic. If the algorithm is required to be deterministic, four robots are also insufficient. This negative result implies that the only way to obtain an optimal algorithm (*w.r.t.* the number of robots participating to the algorithm) is to make use of probabilities, and thus, within the SSYNC model, due to aforementioned impossibility [4].

So, we propose a probabilistic algorithm designed for the SSYNC model that uses four robots to explore all simple tori of size $\ell \times L$, where $7 \leq \ell \leq L$. Hence, in such tori, four robots are necessary and sufficient to solve the (probabilistic) terminating exploration. As a torus can be seen as a 2-dimensional ring, our result shows, perhaps surprisingly, that increasing the number of possible symmetries in the network (due to increasing dimensions) does not necessarily bring an extra cost with respect to the number of robots that are necessary to solve the problem.

Roadmap. Section 2 presents the system model and the problem to be solved. Lower bounds are shown in Section 3. The general solution using four robots is given in Section 4. The correctness of this solution is given in Section 5. Section 6 gives some concluding remarks.

2 Preliminaries

Distributed Systems. We consider systems of autonomous mobile entities called *robots* evolving in a *simple unoriented connected graph* $G = (V, E)$, where V is a finite set of n nodes (*i.e.*, $|V| = n$) and E a finite set of edges, *i.e.*, unordered pairs of distinct nodes. Nodes represent locations that robots can take and edges represent the possibility for a robot to move from one location to another. Two nodes u and v are *neighbors* in G if and only if $\{u, v\} \in E$.

We assume that G is an (ℓ, L)-*Torus* (or a *Torus*, for short), where ℓ and L are two positive integers, *i.e.*, G satisfies the following two conditions: (i) $n = \ell \times L$ and (ii) there exists an order $v_1, \ldots, v_n$ on the nodes of V such that $\forall i \in [1..n]$:

- If $i + \ell \leq n$, then $\{i, i + \ell\} \in E$, else $\{i, (i + \ell) \bmod n\} \in E$.

- If $i \bmod \ell \neq 0$, then $\{i, i+1\} \in E$, else $\{i, i-\ell+1\} \in E$.

Given the previous order $v_1, \ldots, v_n$, for every $j \in [0..(L-1)]$, the sequence $v_{1+j\times\ell}, v_{2+j\times\ell}, \ldots, v_{\ell+j\times\ell}$ is called an ℓ-ring. Similarly, for every $k \in [1..\ell]$, $v_k, v_{k+\ell}, v_{k+2\times\ell}, \ldots, v_{k+(L-1)\times\ell}$ is called an L-ring. Note that when $\ell = L$, any ℓ-ring is also an L-ring and conversely. More generally, we use the term *ring* to arbitrarily designate an ℓ-ring or an L-ring.

Nodes are assumed to be anonymous (they have no access to identifiers or other symmetry breaking capabilities). Moreover, given two neighboring nodes u and v, we assume that there is no explicit or implicit labeling allowing the robots to determine whether u is either on the left, on the right, above, or below v. However, for the purpose of proof description, we may use indices for nodes or robots.

An *isomorphism* of graphs G and H is a bijection f between the vertex sets of G and H such that any two nodes u and v of G are neighbors in G if and only if $f(u)$ and $f(v)$ are neighbors in H. When G and H are one and the same graph, f is called an *automorphism* of G. Remark that an (ℓ, L)-*Torus* and an (L, ℓ)-*Torus* are isomorphic. Hence, as the nodes are anonymous, an (ℓ, L)-*Torus* cannot be distinguished from an (L, ℓ)-*Torus*. So, without loss of generality, we always consider (ℓ, L)-*Tori*, where $\ell \leq L$.

Remark 1 *As an (ℓ, L)-torus is a simple graph, every node has four* distinct *neighbors, and consequently we have: $3 \leq \ell \leq L$ and $n = \ell \times L \geq 9$.*

Robots and Computation. Operating on G are k robots. The robots do not communicate in an explicit way; however they see the position of all other robots in their ego-centered coordinate system and can acquire knowledge from this information.

Each robot operates according to its (local) *program*. We call *protocol* a collection of k *programs*, each one operating on a single robot. Here we assume that robots are *uniform* and *anonymous*, *i.e.*, they all have the same program using no parameter allowing to differentiate them.

The program of a robot consists in executing *Look-Compute-Move cycles* infinitely many times. That is, the robot first observes its environment (Look phase). Based on its observation, a robot then (probabilistically or deterministically) decides to move or stay idle (Compute phase). When a robot decides to move, it moves toward its destination during the Move phase.

During the Compute phase, the decision between moving or staying idle is either deterministic or probabilistic. In the latter case, the robot decides between moving and staying idle using some fixed probability p, with $0 < p < 1$; and we say that the robot *tries to move*.

We assume that robots cannot remember any previous observation nor computation performed in any previous cycle. Such robots are called *oblivious*.

We consider the semi-synchronous model [14]. In this model, time is represented by an infinite sequence of instants 0, 1, 2, ... No robot has access to this global time. At each instant, a non-empty subset of robots is *activated*. Every robot that is activated at instant t *atomically* executes a full cycle between t and $t+1$. Activations are determined by an *adversary*.

Note that in this model, any robot performing a Look operation sees all other robots on nodes and not on edges.

Multiplicity. We assume that during the Look phase, every robot can perceive whether several robots are located on the same node. This ability is called *(global) multiplicity detection*. We shall indicate by $d_i(t)$ the multiplicity of robots present in node v_i at instant t. We consider two versions of multiplicity detection: the *strong* and *weak* multiplicity detections. Under the *weak* multiplicity detection, for every node v_i, d_i is a function $\mathbb{N} \mapsto \{\circ, \bot, \top\}$ defined as follows: $d_i(t)$ is equal to either $\circ$, $\bot$, or $\top$ according to v_i contains none, one or several robots at instant t. If $d_i(t) = \circ$, then we say that v_i is *free* at instant t, otherwise v_i is *occupied* at instant t. If $d_i(t) = \top$, then we say that v_i contains a *tower* at instant t. Under the *strong*

multiplicity detection, for every node v_i, d_i is a function $\mathbb{N} \mapsto \mathbb{N}$ where $d_i(t) = x$ indicates that there are x robots in node v_i at instant t. If $d_i(t) = 0$, then we say that v_i is *free* at instant t, otherwise v_i is *occupied* at instant t. If $d_i(t) > 1$, then we say that v_i contains a *tower (of $d_i(t)$ robots)* at instant t.

Configurations, Views and Executions. To define the notion of *configuration* (of the system), we need to use an arbitrary order $\prec$ on nodes. The system being anonymous, robots do not know this order. Let $v_1, \ldots, v_n$ be the list of the nodes in G ordered by $\prec$. The configuration at instant t is $d_1(t), \ldots, d_n(t)$. We denote by *initial configurations* the configurations from which the system can start at instant 0. Every configuration from which no robot moves or tries to move if activated is said to be *terminal*.

The *view* of robot $\mathcal{R}$ at instant t is a labeled graph isomorphic to G, where every node v_i is labeled by $d_i(t)$, except the node where $\mathcal{R}$ is currently located, this latter node v_j is labeled by $d_j(t), *$. (Indeed, the coordinate system is ego-centered.) Hence, from its view, a robot can compute the view of each other robot, and decide whether some other robots have the same view as its own. The views $\mathcal{V}$ and $\mathcal{V}'$ are *identical* if and only if there exists an isomorphism f of $\mathcal{V}$ and $\mathcal{V}'$ such that every node v of $\mathcal{V}$ has the same label in $\mathcal{V}$ as node $f(v)$ in $\mathcal{V}'$. For example, in Figure 1, the robots located at node v_1 and v_5 respectively have identical views: to see this, apply the isomorphism f such that $f(v_1) = v_5$, $f(v_2) = v_2$, $f(v_3) = v_8$, $f(v_4) = v_4$, $f(v_5) = v_1$, $f(v_6) = v_7$, $f(v_7) = v_6$, $f(v_8) = v_3$, and $f(v_9) = v_9$ on the view of the node located at v_1. Conversely, the view of the node located at v_2 is different of the two others.

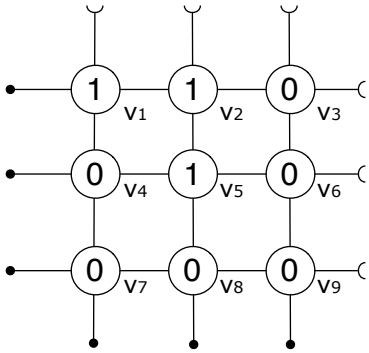

Figure 1: Configuration where two robots have identical views. Numbers inside nodes indicate the strong multiplicity value.

Every decision to move is based on the view obtained during the last Look action. However, it may happen that some edges incident to a node v currently occupied by the deciding robot look identical in its view, *i.e.*, v lies on a symmetric axis of its view. In this case, if the robot decides to take one of these edges, it may take any of them. We assume the worst-case decision in such cases, *i.e.*, the actual edge among the identically looking ones is chosen by the adversary.

Two configurations $d_1, \ldots, d_n$ and $d'_1, \ldots, d'_n$ are *indistinguishable* (resp., *distinguishable* otherwise) if and only if there exists an automorphism on G, $f : V \mapsto V$ such that $\forall i \in \{1, \ldots, n\}$, $d_i = d'_j$ where $v_j = f(v_i)$. For example, Configurations (a) and (b) in Figure 2 are indistinguishable: to see this, apply the automorphism f such that $f(v_1) = v_2$, $f(v_2) = v_5$, $f(v_3) = v_8$, $f(v_4) = v_3$, $f(v_5) = v_6$, $f(v_6) = v_9$, $f(v_7) = v_1$, $f(v_8) = v_4$, and $f(v_9) = v_7$. Conversely, configuration (c) is distinguishable from (a) (resp., (b)).

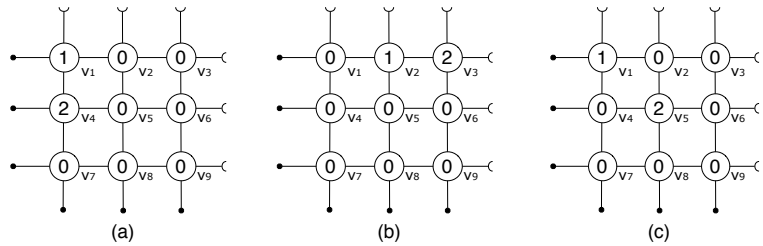

Figure 2: Example of indistinguishable and distinguishable configurations in a (3,3)-torus. Numbers inside nodes indicate the strong multiplicity value.

Consider two indistinguishable configurations γ and γ'. Assume that some robot $\mathcal{R}$ is located at node v_i in γ. Then, by definition, there is some robot $\mathcal{R}'$ located at node $f(v_i)$ in γ'. Moreover, the view of $\mathcal{R}$ in γ is the same as the view of $\mathcal{R}'$ in γ'. So, if $\mathcal{R}$ is activated in γ and may decide to move to some neighboring node v_j, then it is the same for $\mathcal{R}'$: if $\mathcal{R}'$ is activated in γ', it may decide to move to $f(v_j)$. For example, assume that the robot located at v_1 in Configuration (a) of Figure 2 is activated and decides to move toward v_4. Then, there is a possible step where the robot located at node $v_2 = f(v_1)$ in Configuration (b) moves toward $v_3 = f(v_4)$.

A *scheduling* is a list of activation's choices that can be made by the adversary, *i.e.*, a scheduling is any infinite list of non-empty subset of robots $\sigma_0, \sigma_1, \ldots$, where $\forall i \geq 0$, σ_i is the set of robots activated at instant i. An infinite list of configurations $\gamma_0, \gamma_1, \ldots$ can be *generated* from the scheduling $\sigma_0, \sigma_1, \ldots$ if and only if $\forall i \geq 0$, γ_{i+1} can be obtained from γ_i after each robot in σ_i is activated at instant i to atomically perform a cycle (in this case, $\gamma_i\gamma_{i+1}$ is *step*).

We call *execution* any infinite list of configurations $\gamma_0, \gamma_1, \ldots$ that can be *generated* from an arbitrary scheduling and such that γ_0 is a possible initial configuration. An execution e *terminates* if e contains a terminal configuration.

We restrict the power of the adversary by assuming that schedulings are *fair*: a scheduling $\sigma_0, \sigma_1, \ldots$ is *fair* if and only if for every robot $\mathcal{R}$, for every instant i, there exists an instant $j \geq i$ such that $\mathcal{R} \in \sigma_j$. An execution e is *fair* if and only if e can be generated by a fair scheduling.

A particular case of fair scheduling is the *sequential* fair scheduling: a scheduling $\sigma_0, \sigma_1, \ldots$ that is fair and such that $\forall i \geq 0$, $|\sigma_i| = 1$. An execution e is *sequential fair* if it can be generated from a sequential fair scheduling.

Exploration. We consider the *exploration* problem, where k robots, initially placed at different nodes of a graph $G = (V, E)$, collectively explore G before stopping moving forever. By "collectively" we mean that every node of G is eventually visited by at least one robot. More formally, we say that a protocol $\mathcal{P}$ *deterministically* (resp., *probabilistically*) solves the exploration problem assuming a fair scheduling if and only if every fair execution e of $\mathcal{P}$ starting from a *towerless* configuration satisfies: *(1)* e reaches a terminal configuration *in finite time* (resp., *with probability one*[1]), and *(2)* every node is visited by at least one robot during e. Note that the previous definition implies that every initial configuration are *towerless*. Note also that in case of probabilistic exploration, termination is not certain, however the overall probability of non-terminating executions is 0. Observe that the exploration problem is not defined for $k > n$ and is straightforward for $k = n$. (In this latter case, the exploration is already accomplished in the initial towerless configuration.)

[1]We recall that "*with probability one*" has the same meaning as "*almost surely*" or "*almost certainly*," but is different of "*asymptotically almost surely*," see [15], page 186.

3 Lower bound

In this section, we first generalize to arbitrary topologies a result from [4], that was given for rings. We then instantiate this result to obtain a lower bound (actually 4) on the number of robots required to solve (deterministic or probabilistic) exploration in any torus.

To be as general as possible, we assume here that the multiplicity detection of robots is *strong*. Moreover, we consider any (deterministic or probabilistic) exploration protocol $\mathcal{P}$ using a team of k robots in an *arbitrary* topology $G = (V, E)$ of n nodes (*i.e.*, $n = |V|$).

First, assume that $n > k$. Then, the exploration is not (trivially) accomplished in an initial configuration. As robots are oblivious, any terminal configuration of $\mathcal{P}$ in that case should be different from any possible initial configuration. Now, the set of possible initial configurations is exactly the set of all towerless configurations. So, any terminal configuration necessarily contains a tower:

Remark 2 *If $n > k$, any terminal configuration of $\mathcal{P}$ contains at least one tower.*

Consider an instant t. Despite a non-empty subset of robots being activated at instant t, it is possible that no robot moves between t and $t + 1$. Such "empty" steps being useless, we may consider the *minimal relevant subsequence* of the execution defined below, where all "empty" steps have been removed:

Definition 1 (MRS) *Let s be a sequence of configurations. The* minimal relevant subsequence *of s, noted $\mathcal{MRS}(s)$, is the maximal subsequence of s where no two consecutive configurations are identical.*

Lemma 1 *Assume that $n > k$. Let e be any sequential fair execution e of $\mathcal{P}$ that terminates,*[2] *$\mathcal{MRS}(e)$ has at least $n - k + 1$ configurations containing a tower of less than k robots.*

Proof. Assume, by the contradiction, that there is a sequential fair execution e of $\mathcal{P}$ that terminates and such that $\mathcal{MRS}(e)$ has less than $n - k + 1$ configurations containing a tower of less than k robots.

Take the last configuration α without tower that appears in e and all remaining configurations that follow in e (all of them contains a tower) and form e' (e' necessarily exists, by Remark 2). As α could be an initial configuration and e is a sequential fair execution that terminates, e' is also a sequential fair execution of $\mathcal{P}$ that terminates.

By definition, $\mathcal{MRS}(e')$ is constituted of a configuration with no tower only followed by configurations with tower and $n-k$ new nodes (remember that k nodes are already visited in the initial configuration) must be visited before e' reaches its terminal configuration.

Consider a step $\beta\beta'$ in e'.

1. If $\beta = \beta'$, then no node is visited during the step.
2. If $\beta \neq \beta'$, then there are three possible cases:
   (a) *β contains no towers.* In this case, $\beta = \alpha$ (the initial configuration of e') and β' contains a tower. As only one robot moves in $\beta\beta'$ to create a tower (e' is sequential), no node is visited during this step.
   (b) *β contains a tower and β' contains a tower of k robots.* As e' is sequential and all robots are located at the same node in β', one robot moves to an already occupied node in $\beta\beta'$ and no node is visited during this step.

[2] As explained in the model, in case where $\mathcal{P}$ is probabilistic, some of its executions may not terminate, even if the overall probability that such executions occur is 0.

(c) *β contains a tower and β' contains a tower of less than k robots.* In this case, at most one node is visited in $\beta\beta'$ because e' is sequential.

To sum up, only the steps from a configuration containing a tower to a configuration containing a tower of less than k robots (Case 2.(c)) allow to visit at most one node each time. Now, in $\mathcal{MRS}(e')$ there are less than $n-k+1$ configurations containing a tower of less than k robots and the first of these configurations appearing into e' is consecutive to a step starting from the initial configuration (Case 2.(a)). Hence, less than $n-k$ nodes are dynamically visited during e' and, as exactly k nodes are visited in the initial configuration, less than n nodes are visited when e' terminates, a contradiction. □

Lemma 2 *If $n > k$, then for every sequential fair execution e of $\mathcal{P}$ that terminates, $\mathcal{MRS}(e)$ has at least $n-k+1$ configurations containing a tower of less than k robots and any two of them are distinguishable.*

Proof. Consider any sequential fair execution $e = \gamma_0, \gamma_1, \ldots$ of $\mathcal{P}$ that terminates. As e terminates, $\mathcal{MRS}(e)$ has a finite number of configurations. So, let x be the number of configurations in $\mathcal{MRS}(e)$ containing a tower of less than k robots. By Lemma 1, $x \geq n-k+1$.

We first show that (*) *if $\mathcal{MRS}(e)$ contains at least two configurations having a tower of less than k robots that are indistinguishable, then there exists a sequential fair execution e' that terminates and such that $\mathcal{MRS}(e')$ has x' configurations containing a tower of less than k robots, where $x' < x$.*

Assume that there are two indistinguishable configurations $\gamma_t = d_1^t, \ldots, d_n^t$ and $\gamma_{t'} = d_1^{t'}, \ldots, d_n^{t'}$ in $\mathcal{MRS}(e)$ having a tower of less than k robots. Without loss of generality, assume that $t' > t$. By definition, there exists an automorphism f on G such that $\forall i \in \{1, \ldots, n\}$, $d_i^t = d_j^{t'}$ where $v_j = f(v_i)$. Then, $e' = \gamma_0, \ldots, \gamma_t, \gamma'_{t'+1}, \gamma'_{t'+2}, \ldots$ is a sequential fair execution of $\mathcal{P}$, where that $\forall z \geq t'+1$, we have $\gamma'_z = d^z_{g(1)}, \ldots, d^z_{g(n)}$ where g is a bijection such that $\forall s \in [1..n]$, $f(v_s) = v_{g(s)}$ and $\gamma_z = d_1^z, \ldots, d_n^z$. Moreover, $\mathcal{MRS}(e')$ has x' configurations containing a tower of less than k robots, where $x' < x$.

By (*), if $\mathcal{MRS}(z)$ contains less than $n-k+1$ distinguishable configurations altogether with a tower of less than k robots, it is possible to (recursively) construct a sequential fair execution e' of $\mathcal{P}$ that terminates such that $\mathcal{MRS}(e')$ has less than $n-k+1$ configurations containing a tower of less than k robots, a contradiction to Lemma 1. Hence, the lemma holds. □

From Lemma 2, we can deduce the following corollary:

Corollary 1 *Considering any (probabilistic or deterministic) exploration protocol for k robots on a graph of $n > k$ nodes working under any fair scheduling, there exists a set $\mathcal{S}$ of at least $n-k+1$ configurations such that:*

1. *any two different configurations in $\mathcal{S}$ are distinguishable, and*
2. *in every configuration in $\mathcal{S}$, there is a tower of less than k robots.*

Theorem 1 *Assuming fair schedulings, $\forall k, 0 \leq k < 3$, there is no protocol to (deterministically or probabilistically) explore any torus of n nodes using k robots.*

Proof. First, $k < 9 \leq n = \ell \times L$ (see Remark 1). Then, for $k = 0$, the theorem is trivially verified. Consider now the case $k = 1$ and $k = 2$. With one robot it is impossible to construct a configuration with a tower; and with two robots it is impossible to construct a configuration with a tower of less than k robots ($k = 2$). Hence, for $k = 1$ and $k = 2$, the theorem is a direct consequence of Corollary 1. □

The previous theorem excludes that $\mathcal{P}$ works with $k < 3$. Let now assume that $k = 3$ and consider any arbitrary (ℓ, L)-torus (remember that $n = \ell \times L \geq 9$, Remark 1). Then, by Corollary 1, we should be able to exhibit a set $\mathcal{S}$ of $n-2$ configurations such that:

1. Any two different configurations in $\mathcal{S}$ are distinguishable, and

2. In every configuration in $\mathcal{S}$, there is a tower of 2 robots.

Such configurations differ according to the relative positions of the tower and the robot which is alone. Two cases are then possible depending on whether $\ell = L$ or $\ell < L$. In the former case, the size of $\mathcal{S}$ is bounded by $\sum_{i=2}^{\lfloor \frac{L}{2} \rfloor + 1} i = \frac{\lfloor \frac{L}{2} \rfloor \times (\lfloor \frac{L}{2} \rfloor + 3)}{2}$. For example (refer to Figure 3), in a $(5, 5)$-torus, the size of $\mathcal{S}$ is at most 5.

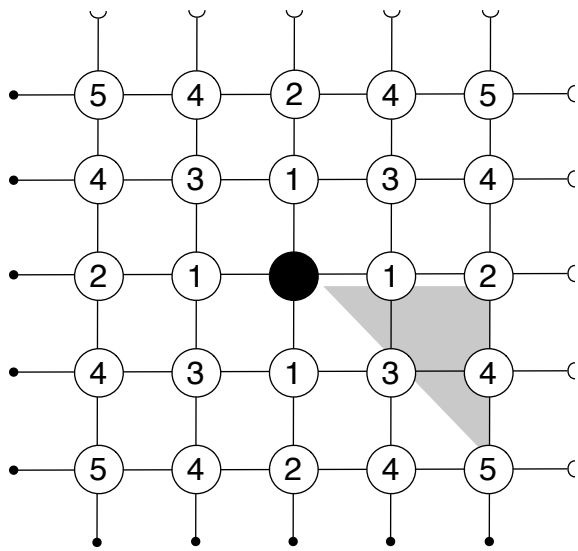

Figure 3: For every value i inside a white node, every two configurations where (1) the black node contains the tower of two robots and (2) any white node of number i contains the single robot are indistinguishable.

In the latter case, the size of $\mathcal{S}$ is bounded by $(\lfloor \frac{\ell}{2} \rfloor + 1)(\lfloor \frac{L}{2} \rfloor + 1) - 1$. For example (refer to Figure 4), in a $(5, 6)$-torus, the size of $\mathcal{S}$ is at most 11.

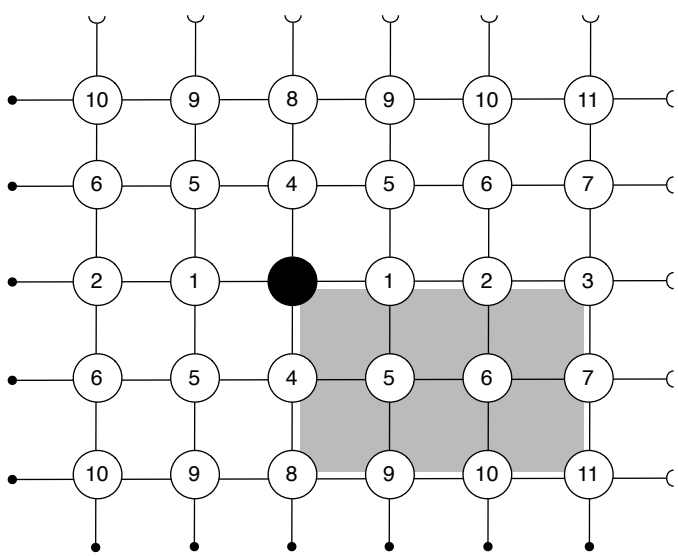

Figure 4: For every value i inside a white node, every two configurations where (1) the black node contains the tower of two robots and (2) any white node of number i contains the single robot are indistinguishable.

Let now study the case where $\ell = L$. Then, $\frac{\lfloor \frac{L}{2} \rfloor \times (\lfloor \frac{L}{2} \rfloor + 3)}{2}$ should be greater or equal to $n-2$, *i.e.*, $L^2 - 2$.

$$\begin{array}{lcl}
\frac{\lfloor \frac{L}{2} \rfloor \times (\lfloor \frac{L}{2} \rfloor + 3)}{2} & \geq & L^2 - 2 \\
\lfloor \frac{L}{2} \rfloor \times (\lfloor \frac{L}{2} \rfloor + 3) & \geq & 2L^2 - 4 \\
\frac{L^2}{4} + \frac{3L}{2} & \geq & 2L^2 - 4 \\
\frac{L^2+6L}{4} & \geq & 2L^2 - 4 \\
L^2 + 6L & \geq & 8L^2 - 16 \\
7L^2 - 6L - 16 & \leq & 0
\end{array}$$

$\Delta = 484 > 0$, so $7L^2 - 6L - 16 = 0$ has two solutions: $\frac{6-\sqrt{484}}{14}$ and $\frac{6+\sqrt{484}}{14}$; and $7L^2 - 6L - 16 \leq 0$ for $L \in [\frac{6-\sqrt{484}}{14}; \frac{6+\sqrt{484}}{14}]$. Now $L \geq 3$ (Remark 1) and $\frac{6+\sqrt{484}}{14} = 2$. So, we obtain a contradiction: there is neither probabilistic nor deterministic exploration protocol in that case, even assuming a fair scheduling.

Let now study the case where $\ell < L$. Then, $(\lfloor \frac{\ell}{2} \rfloor + 1)(\lfloor \frac{L}{2} \rfloor + 1) - 1$ should be greater or equal to $n-2$, *i.e.*, $\ell \times L - 2$.

$$(\lfloor \tfrac{\ell}{2} \rfloor + 1)(\lfloor \tfrac{L}{2} \rfloor + 1) - 1 \geq \ell \times L - 2$$

$$\frac{\ell \times L}{4} + \frac{\ell}{2} + \frac{L}{2} + 1 \geq \ell \times L - 1$$

$$\frac{\ell \times L + 2\ell + 2L}{4} \geq \ell \times L - 2$$

$$\ell \times L + 2\ell + 2L \geq 4\ell \times L - 8$$

$$2\ell + 2L + 8 \geq 3\ell \times L$$

Now, as $3 \leq \ell < L$ (Remark 1), $2\ell + 2L + 8 \geq 3\ell \times L$ has no solution: there is neither probabilistic nor deterministic exploration protocol in that case, even assuming a fair scheduling.

Hence, there is neither probabilistic nor deterministic protocol to explore any torus with 3 robots and, with Theorem 1, we can conclude:

Theorem 2 *Assuming fair schedulings, $\forall k, 0 \leq k < 4$, there is no protocol to (deterministically or probabilistically) explore any torus of n nodes using k robots.*

Consider now the *deterministic* exploration with $k = 4$ robots. Assume any (ℓ, L)-*Torus* such that $\ell = L$ and ℓ is even. Then, it is possible to initially place the four robots in such way that they have all identical views and all their possible destinations looked identical (just form a square whose adjacent sides have length $\frac{\ell}{2}$). In this case, the adversary can choose to synchronously activate all robots at each step in such way that the initial symmetry continues: we obtain a fair execution which does not terminate. Hence, following the idea of [16], we have:

Theorem 3 *Assuming fair schedulings, $\forall k, 0 \leq k \leq 4$, there is no protocol to* deterministically *explore all tori of n nodes using k robots.*

Notice that the previous impossibility result can be circumvented, for example, by making restrictions on possible initial configurations [10].

4 Optimal Algorithm

We propose in Algorithm 1 a probabilistic algorithm to explore with 4 robots any (ℓ, L)-torus such that $7 \leq \ell \leq L$, assuming weak multiplicity detection. Before providing informal explanations, we first need to define some terms.

4.1 Definitions

We propose a probabilistic algorithm to explore with 4 robots any (ℓ, L)-torus such that $7 \leq \ell \leq L$, assuming weak multiplicity detection. Before providing informal explanations, we first need to define some terms.

Let v_1 and v_2 be two nodes containing robot r_1 and r_2, resp. r_1 is a neighboring robot of r_2, and conversely. A *block* is a maximal elementary path along some ring of the torus $B = u_i, u_{i+1}, \ldots, u_{i+m}$ with $m > 0$, where each node is occupied by exactly one robot. A robot that does not belong to a block is said to be *isolated*. A *hole* is any maximal non-empty elementary path of free nodes $H = u_i, u_{i+1}, \ldots, u_{i+m}$ that is along some ring of the torus. The *size* of a block (resp., a hole) is the number of nodes it contains. A block (resp. a hole) of size x is said to be an x-*block* (resp., a x-hole). Given the block B (resp., the hole H), the nodes u_i and u_{i+m} are termed as the *extremities* of B (resp., H). We call *neighbor* of a hole (resp. a block) any node that does not belong to the hole (resp. the block) but is neighbor of one of its nodes. In this case, we also say that the hole (resp. the block) is a *neighboring hole* (resp. *neighboring block*) of the node. By extension, any robot that is located at a neighboring node of a hole (resp. a block) is also referred to as a neighbor of the hole (resp. the block). A node u is said to be *safe* if there is at most one robot that is located within distance one from u. We call *Couple* any ℓ-ring that contains exactly two robots.

4.2 Overview of the algorithm

Our algorithm works in three distinct successive phases, respectively called **SetUp**, **Tower**, and **Exploration**.

Starting from any towerless configuration, the aim of the **SetUp Phase** is to arrange the robots in such a way that they eventually form a $\Diamond$.*Configuration* (see Figure 5), without creating any tower during the process. This first phase is probabilistic. A $\Diamond$.Configuration is a configuration, where (1) there are two ℓ-rings of the torus that both contain a 2-block, and (2) there are two robots that have two robots in their neighborhood.

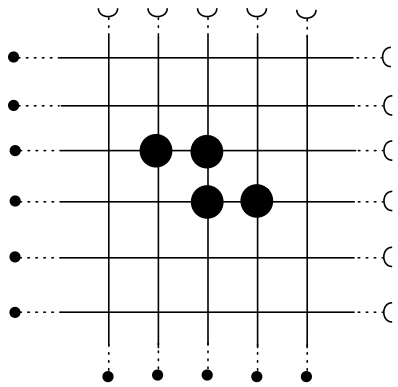

Figure 5: A $\diamondsuit$.Configuration.

Once a $\diamondsuit$.Configuration is built, **Tower Phase** begins. This phase is also probabilistic and consists in creating a tower using the two neighboring robots that have exactly two robots in their neighborhood. Once the tower is created, the location of robots give an explicit orientation to the torus; and the last phase, **Exploration Phase**, begins. This phase is deterministic. The two isolated robots collaborate together to deterministically explore the torus and eventually stop.

Algorithm 1 Main Program.

```
1.01  if the configuration contains no tower and is not a ◇.Configuration
1.02  then  execute Procedure Phase SetUp
1.03  else  if the configuration is a ◇.Configuration
1.04        then  execute Procedure Phase Tower
1.05        else  execute Procedure Phase Exploration
```

We now explain the three successive phases in more details.

4.3 Phase SetUp

Phase SetUp is formally described in Algorithms 2-7.

4.3.1 Configurations

The behavior of Phase SetUp is led by the configurations. Below, we define the main classes of configurations.

A configuration is said to be a *Double-Trap1* (see Figure 6) if there exists an ℓ-ring R that contains a 3-block having exactly one extremity with a neighboring robot that is not in R.

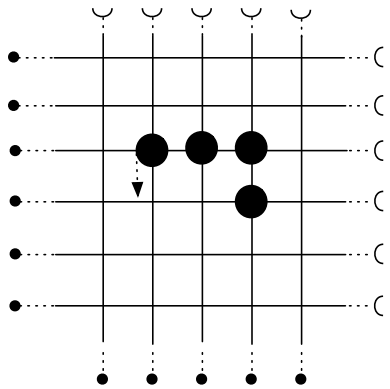

Figure 6: *Double-Trap1* Configuration.

A configuration is a *Double-Trap2* (see Figure 7) if there is one isolated robot at some node z and two 2-blocks $B1 = u, v$ and $B2 = x, y$ such that (1) $v = x$, (2) $B1$ is on a ℓ-ring, (3) z and y are on a ℓ-ring parallel to the one containing $B1$, and (4) z is at distance 2 of both u and y.

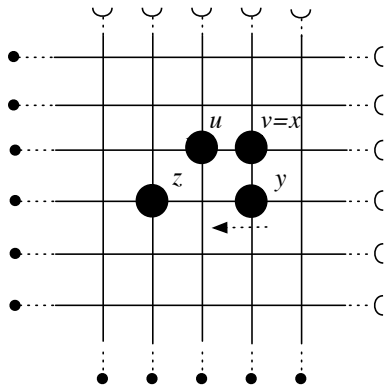

Figure 7: *Double-Trap2* Configuration.

A configuration C is said to be *Regular* if C contains no tower, C is not a $\Diamond$.configuration, and the robots can be split in two pairs $\{r_1, r_2\}$ and $\{r_3, r_4\}$ such that the views of r_1 and r_2 (resp. the views of r_3 and r_4) are identical. (A particular case of configuration Regular is a configuration where all robots have identical views.) A configuration C is said to be *Triplet* if C is not a Double-Trap1 and there is a ℓ-ring R that contains exactly 3 robots. When R contains neither a 3-block nor a 2-block, we define the *Wall* as the ring perpendicular to R that contains the robot not in R, see Figure 8.

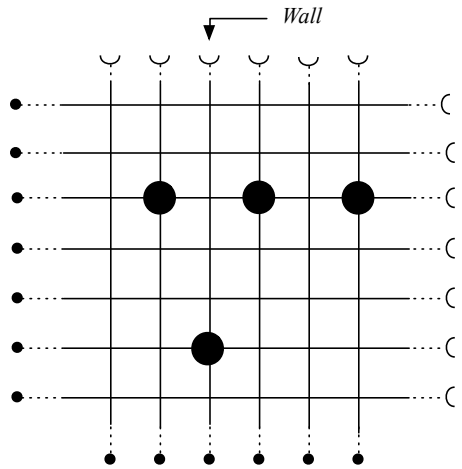

Figure 8: A wall in a *Triplet* Configuration.

A configuration C is said to be *Twin* (see Figure 9) if C contains a couple, but is neither Double-Trap1, nor Double-Trap2, nor $\Diamond$, nor Regular, not Triplet.

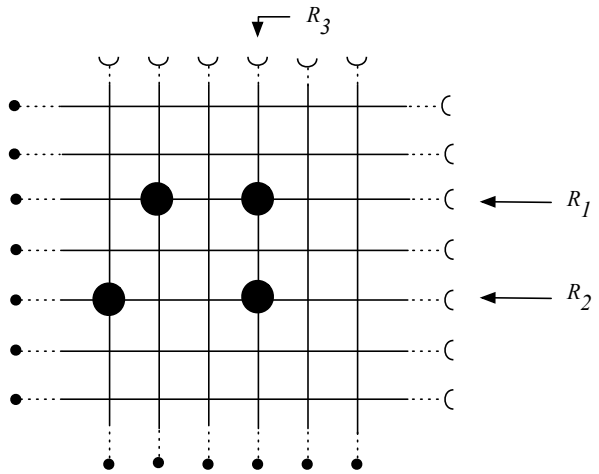

Figure 9: *Twin* Configuration

.

A configuration C is said to be *Isolated* if C is not Regular and there exists at most one robot on each ring of size ℓ. Finally, a configuration C is said to be *Quadruplet* if C is not Regular and there exists an ℓ-ring R that contains 4 robots.

4.3.2 Description of Phase SetUp

Recall that the aim of the SetUp phase is to create a $\Diamond$.Configuration, starting from an arbitrary towerless configuration and without creating any tower during the phase. As a matter fact, we try to make robots move deterministically as much as possible. However, there are symmetric configurations that require robots to move probabilistically. The main one is the following:

(a) *The configuration is of type Regular.* To break this symmetry, each robot tries to move to a safe node. Doing so, the symmetry is broken without creating tower after one step with positive probability.

So, with probability one, the system eventually leaves Case (a) to a configuration that matches one of the following cases:

(b) The configuration is of type *Double-Trap2* or *Double-Trap1*. In the former case, by moving one robot as shown in Figure 7, the system reaches a $\Diamond$.Configuration. In the latter case, we easily obtain a *Double-Trap2* by moving only one robot as shown in Figure 6.

(c) The configuration is of type *Triplet*. In this case, there are three robots in the same ℓ-ring R and we deterministically build a Double-Trap1 configuration. There are several cases to consider. (i) Three robots on R already forms a 3-block. Then, the remaining robot moves to the adequate position to build a *Double-Trap1*. (ii) R contains a 2-block. Then, the robot in R that is not part of the 2-block moves to create the 3-block. Otherwise (iii), a *Wall* is defined and we use it to create a 2- or 3-block on R as follows: If a node v that intersects both the Wall and R is occupied by some robot, the two other robots on R move towards the Wall to create a 2- or a 3-block (depending on the choices of the scheduler). Otherwise, each robot of R that is not neighbor of v moves towards v until a 2-block is created on R.

(d) The configuration is of type *Twin*. In this case, the aim is to reach a Triplet configuration with positive probability. Assume first that the configuration contains only one couple. Then, the two robots that are not part of the couple compete together (using try to move) so that eventually one of them is closer from the couple than the other. Then, the closest one moves to create the Triplet configuration.
Otherwise, the configuration may also contain two or three couples, but not four, otherwise we would be in Case (a). If the configuration contains two couples, then each robot that is neighbor of a safe node outside any couple tries to move to that safe node. Then, with a positive probability, only one of them moves and we retrieve the previous case where there is only one couple. If the configuration contains three couples, then $\ell = L$ and two robots belong to two couples. These robots try to move to a safe neighboring node. With positive probability, only one of them moves, and we retrieve a previous case: a Twin configuration with two couples.

In all remaining configurations, the aim is to reach either a *Triplet* or a *Twin*.

(e) The configuration is of type *Quadruplet*. In such a configuration, the four robots belong to some ℓ-ring R. We then consider the subgraph G_R induced by the nodes of R. This graph is isomorphic to an elementary cycle. Now, as not all robots have identical views, we can discriminate either one unique robot or two robots (using the sizes of the holes in G_R). We let those robots move outside R. Hence, the system reaches either a Twin or a Triplet configuration.

(f) The configuration is of type *Isolated*. We consider two subcases:

- $\ell < L$. In this case, we discriminate L-rings according to the number of robots on them.
  When some but not all robots are alone in their L-rings, they move along their L-ring to eventually form a Triplet or a Twin configuration with the blocked ones.
  In the case where there are two L-rings that contain 2 robots, robots try to move to a neighboring safe node (if any) outside the L-ring they belong to. Hence, without positive probability, we retrieve the previous case.
  In the case where every L-ring contains at most one robot. We first make robots probabilistically move to discriminate a unique smallest rectangle that encloses the 4 robots. When there are several possible smallest enclosing rectangles (SER), we decrease their number by proceeding as follows: if a robot r that is neighbor of a safe node u such that if it moves to u and it is the only one to move, the number of SER decreases, then r tries to move to u. In this case, with positive probability, only one robot moves, reducing the number of possible smallest enclosing rectangles or leading directly the system to a Twin configuration. If we have a unique smallest rectangle, s, that encloses the 4 robots and the configuration is not Twin, we discriminate the robots according to their place in s: at a corner, on a side, or inside of s. This allows us to block some of them. The other ones move (or try to move, in case of possible symmetry) to create

either a Triplet or a Twin configuration.
Finally, in the case where there is a single L-ring R that contains the four robots, we proceed as in Case (e), but this time we operate on a L-ring: one or two robots eventually leave the L-ring and we retrieve one of the previous cases.

– $\ell = L$. We probabilistically discriminate, as previously, a unique smallest rectangle that encloses the 4 robots. Once the system has reached a configuration containing a unique smallest enclosing rectangle, we proceed as in case $\ell < L$. However, note that this case is simpler than the previous one because all rings have the same size (ℓ), while in the previous case we had to take care that robots gather on a "small" ring.

We will prove the convergence with probability one from any towerless configuration to a $\Diamond$.Configuration (without creating any tower during the process) by showing that the transitions given in Figure 10 can be made in finite number of steps and with positive probability. Notice that some of them are made deterministically. Note also that there are many other possible transitions.

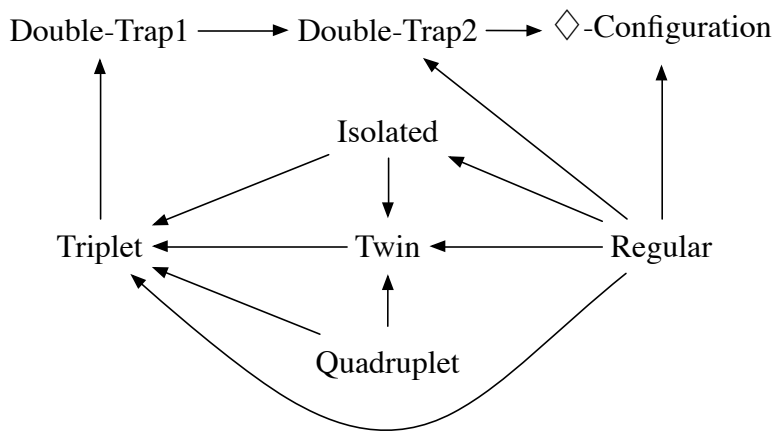

Figure 10: Possible transitions during SetUp.

Algorithm 2 SetUp.

```
Let C be the current configuration
if C is Regular
then  try to move to a safe node
else  if C is Double-Trap1
      then  Let R be the ring containing the 3-block
            if I am the extremity of the 3-block that has only one neighboring robot
            then move to the ring parallel to R that contains a single robot
      else  if C is Double-Trap2
            then  if I have a neighboring robot and there is a distance-2 robot r on the same ring
                  then  move towards r
            else  if C is Quadruplet
                  then  execute Procedure Quadruplet
                  else  if C is Triplet
                        then  execute Procedure ◇-Creation
                        else  if C is Isolated
                              then  if L = ℓ
                                    then  execute procedure Isolated-1
                                    else  execute procedure Isolated-2
                              else  if C is Twin
                                    then  execute Procedure Twin
```

Algorithm 3 Procedure $\Diamond$-Creation.

```
Let R be the ring of the torus that contains 3 robots
if R contains no 3-block
then  if R contains no 2-block
      then  if R ∩ Wall= Free-Node
            then  if I am on R, but not neighbor of Free-Node
                  then  move along R towards Free-Node
            else  // R ∩ Wall= Occupied-Node
                  if I am not on the Wall
                  then  move towards the Wall
      else  // R contains a 2-block
            if I am on R, but not part of the 2-block
            then  move towards the 2-block
else  // R contains a 3-block
      if I am not part of R
      then  move towards a free node that closest from an extremity of the 3-block and not part of R
```

Algorithm 4 Procedure Twin.

```
if there is a unique Couple C
then  if there is a unique robot r that is outside C and closest to C
      then  if I am r
            then  move to an adjacent free node on a shortest path to a free node of C
      else  if I am outside the Couple
            then  let R1 be my ring perpendicular to C
                  let R2 be the ring parallel to C which contains the other robot outside C
                  try to move to a neighboring free node on R1 which does not belong to C or R2
else // there are two or three Couples
      if there are two Couples
            if I am neighbor of a safe node u outside any couple
            then  try to move to u
      else  // there are three Couples
            if I belong to two Couples
            then  try to move to a neighboring safe node
```

Algorithm 5 Procedure Quadruplet.

```
We consider the subgraph G_R induced by the nodes of the ring R that contains the 4 robots
(G_R is isomorphic to a cycle)
if G_R contains exactly one smallest hole
then  if I am neighbor to such a hole
      then  move to an adjacent free node outside R
else  if G_R contains exactly one biggest hole
      then  if I am neighbor to such a hole
            then  move to an adjacent free node outside R
      else  if I am neighbor to the two smallest hole
            then  move to an adjacent free node outside R
```

Algorithm 6 Procedure Isolated-1 (Case $\ell = L$).

```
6.01 if there are several possible smallest enclosing rectangles
6.02 then  if I am neighbor of a safe node u such that if I move to u and I am the only one to move, the number of SER decreases
6.03       then try to move to u
6.04 else
6.05       Let s the unique smallest rectangle that encloses the four robots
6.06       We call internal robot any robot that is not on any side of s
6.07       if there is exactly one robot that is at a corner c of s
6.08       then  if I am internal
6.09             then  move towards the closest free node on a side of s having c as extremity
6.10       else  if there are exactly two robots at two corners c1, c2 of s
6.11             then  if I am a closest internal robot to an occupied corner of s
6.12                   then  move towards the closest free node on a side of s having c1 or c2 as extremity
6.13             else  // there is no robot of any corner of s
6.14                   Let d be the minimum distance between any occupied node and a corner of s
6.15                   if there is a unique robot r that is at distance d from a corner of s
6.16                   then  if I am r
6.17                         then  move towards the closest corner of s
6.18                   else  if there are two robots r1, r2 that are at distance d from the same corner of s
6.19                         then  if I am neither r1 nor r2
6.20                               then move towards the corner that is common to my side of s
6.21                               and the side of s where r1 or r2 are
6.22                         else  if I am at distance d from a corner of s
6.23                               then  move towards the closest corner of s
```

Algorithm 7 Procedure Isolated-2 (Case $\ell < L$).

```
7.01 if every L-ring contains at most one robot
7.02 then  Let r1, r2, r3, and r4 be the four robots
7.03       if there are several possible smallest enclosing rectangles
7.04       then  if I am neighbor of a safe node u such that if I move to u and I am the only one to move, the number of SER decreases
7.05             then try to move to u
7.06       else
7.07             Let s the unique smallest rectangle that encloses the four robots
7.08             There are two robots, say r1 and r2, which are on two parallel sides of s that are on ℓ-rings
7.09             if I am r3 or r4
7.10             then  move along my L-ring toward the closest ℓ-ring which is a side of s
7.11 else  if there is exactly one L-ring R that contains 2 or 3 robots
7.12       then  if I am alone in my L-ring
7.13             then  move along my L-ring towards a closest free node having a node of R in its ℓ-ring
7.14       else  if there are two L-rings that contain 2 robots
7.15             then  if there is a safe neighboring node u outside my current L-ring
7.16                   then  try to move to u
7.17             else  // there is one L-ring R that contains all robots
7.18                   Let G_R be the subgraph induced by R
7.19                   Apply Procedure Quadruplet on G_R
```

4.4 Tower Phase

This phase starts from any $\Diamond$.Configuration, see for example Figure 5. Let u_1 and u_2 be the two occupied neighboring nodes having themselves two occupied neighboring nodes. Let r_1 and r_2 be the two robots located at u_1 and u_2, respectively. During this phase, r_1 and r_2 *try to move* towards each other anytime they are activated; the other two robots do not move if activated. If only one of them is activated, a tower is created. If both are activated simultaneously, there is a positive probability that only one of those moves; otherwise the system remains in a $\Diamond$.Configuration. As the scheduler is fair, both of them are activated regularly until a tower is created. So, this latter event eventually occurs with probability one. Therefore, a tower $\mathtt{T}$ is created with probability one on either u_1 or u_2 — refer to Figure 11 — and the Tower phase is finished.

Algorithm 8 Phase Tower.

8.01 **if** I am neighbor of two occupied nodes
8.02 **then** Try to move to my neighbor that has also two neighboring occupied nodes

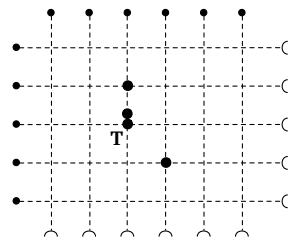

Figure 11: Initial Configuration for Deterministic Exploration.

4.5 Phase Exploration

We first need some definitions. Given two nodes u and v, let R_{uv} be a *smallest enclosing rectangle* that includes both u and v. Let α_{uv} (β_{uv}) be the length in terms of hops of one of the smallest (resp., greatest) side of R_{uv}, R_{uv} is an $(\alpha_{uv}, \beta_{uv})$-rectangle. The (Manhattan) *distance* between two nodes u and v, denoted by d_{uv}, is equal to $\alpha_{uv} + \beta_{uv}$. We define a total order on distances as follows: given four nodes u, v, u', and v', $d_{uv} \leq d_{u'v'}$ *iff* either $d_{uv} < d_{u'v'}$ or $d_{uv} = d_{u'v'}$ and $\beta_{uv} \leq \beta_{u'v'}$.

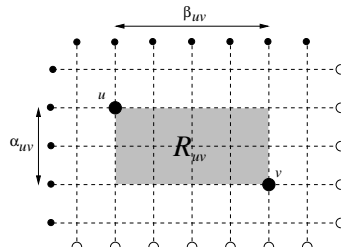

Figure 12: Smallest Enclosing Rectangle.

The deterministic exploration starts assuming that two robots form a tower. Denote the node holding the tower by T. The two rings passing through T are called *coordinate rings*. In the sequel, '$\circ$' (respectively, '$\star$') denotes the *nearest* (respectively, *farthest*) single node (or robot) from T, *i.e.,* $d_{\circ T} < d_{\star T}$. Note that our algorithm ensures that both '$\circ$' and '$\star$' remain the same robots until the end of the exploration. Given a node u, if $\alpha_{Tu} < \beta_{Tu}$ and $\{\alpha_{Tu}, \beta_{Tu}\} \neq \{\lfloor \frac{\ell}{2} \rfloor, \lfloor \frac{L}{2} \rfloor\}$, then there exists an orientation of the coordinate rings such that $u = (\alpha_{Tu}, \beta_{Tu})$, see for example Figure 13. In the following, when it is possible, we build a coordinate system over R_{Tu} by setting the x-axis (the y-axis) as the coordinate rings that is parallel to the smallest (resp., greatest) side of $R_{T,*}$ and by orienting both axis so that the coordinates of u are positive.

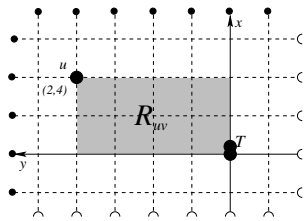

Figure 13: Coordinate System.

The formal algorithm in shown in Algorithms 9 to 14. The main idea of Phase Exploration is the following: both robots that are not part of the tower collaborate together in order to explore the whole torus. They alternate between two roles: *Explorer* and *Leader*. Leader $\mathcal{L}$ allows to build a coordinate system $S^{\mathcal{L}}$ over $R_{\mathtt{T}\mathcal{L}}$. The explorer is in charge of deterministically exploring the torus over $S^{\mathcal{L}}$. The exploration works in three phases, executed in sequence:

Phase 1: Figure 14 illustrates that phase. Robot $\star$ (*i.e.,* the farthest robot of $\mathtt{T}$) plays the Leader role. Starting from the configuration built by Phase Tower, Robot $\star$ first built a $(1, 2)$-rectangle with $\mathtt{T}$ by moving in the opposite direction to $\circ$, see Moving #1 in Line 9.03. $R_{\mathtt{T}*}$ allows to build a coordinate system S^*, where Robot $\star$ occupies Node $(1, 2)$ *w.r.t.* S^*. Then, Robot $\circ$ initiates a spiral-shaped exploration. It visits the nodes that form the first surrounding square around $\mathtt{T}$ and stops at node $(-1, -1)$, see Moving #2 in Lines 9.04-9.10 and Procedure $1stSpiral(phase)$ in Algorithm 10.

Next, Robot $\star$ moves to node $(2, 3)$ passing through node $(1, 3)$, see Movings $3'$ and $3''$ in Lines 9.08 and 9.14. Then, Robot $\circ$ visits the nodes that form the second surrounding square around $\mathtt{T}$ and stops at $(-2, -2)$, see Moving #4 in Lines 9.26. Finally, Robot $\star$ moves back to $(1, 3)$, followed by Robot $\circ$ that moves back to $(-2, -1)$, see Movings #5 and #6 in Lines 9.24 and 9.17. This ends the first phase.

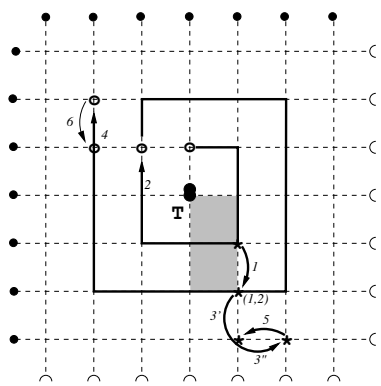

Figure 14: First phase of exploration.

Note that our method requires that Robot $\star$ must be able to move at least three lines away from the tower. Furthermore, Robot $\circ$ must be able to visit the two squares centered on the tower and the orientation built by Robot $\star$ must be unambiguous. These three conditions constrain the torus to be of size at least 7×7.

Phase 2: In this phase, Robot $\circ$ is the leader. $R_{\mathtt{T}o}$ provides a coordinate system S°, where Robot $\circ$ is located at $(1, 2)$. Robot $\star$ now proceeds to the spiral exploration by visiting surrounding squares around $\mathtt{T}$ one after another, see Figures 15 and 16. This process is initiated at Line 9.20 and managed by Procedures $2ndSpiralMngmt$ and $2ndSpiral$ (Algorithms 11 and 12). Robot $\star$ first explores the third surrounding square around $\mathtt{T}$, then the fourth, and so forth, until it visits the $(\lfloor \frac{\ell}{2} \rfloor - 1)$-th square.

Then, there are two cases depending on the parity of ℓ: If ℓ is odd, then Robot $\star$ visits the whole $\lfloor \frac{\ell}{2} \rfloor$-th square and finish at the negative (*w.r.t* S°) corner of the square, see for example Figure 15. Otherwise (ℓ is even), Robot $\star$ visits half of the $\lfloor \frac{\ell}{2} \rfloor$-th square only and stops at the positive corner (*w.r.t.* S°) of the square, see for example Figure 15. In both cases, if $\ell = L$, then the exploration is done, see Line 11.10.

Phase 3: This last phase is performed only if $\ell \neq L$. In that case, Robot $\star$ terminates the exploration by going alternatively from the left to the right and from the right to the left among the nodes forming the remaining of the rectangle. If ℓ is odd, then Robot $\star$ progresses towards the negative (*w.r.t.* S°) side

of the torus, see Figure 15 and Algorithm 13. Otherwise (ℓ is even), the progression is made on the positive side, see Figure 16 and Algorithm 14. In both cases, the exploration ends either on the positive side or the negative side of the L-th line, depending on either L is odd or even, see Figures 15-16 that shows two of the four possible cases of termination depending on the parity of both ℓ and L.

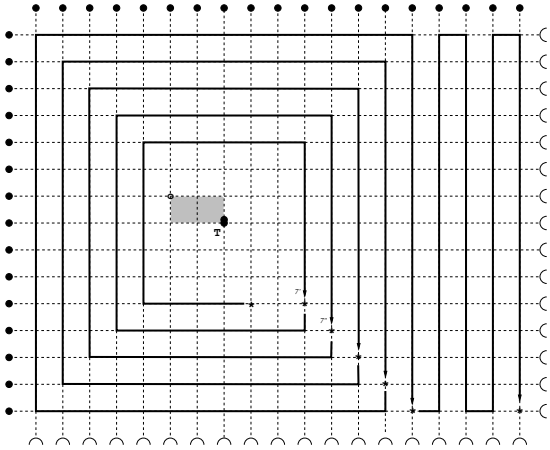

Figure 15: Second and third phase of exploration, odd case.

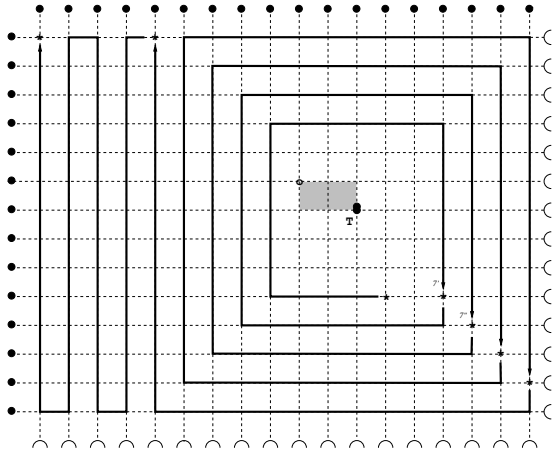

Figure 16: Second and third phase of exploration, even case.

Algorithm 9 Phase Exploration, Main Program.

```
if R_T* is a (1,1)-rectangle
then  if I am ⋆
      then Move in the opposite direction than the direction provided by R_To //Fig. 14, Moving #1
else  Let S* be the coordinate system built over R_T*
      if R_T* is an (1,2)-rectangle
      then  if ∘ = (−1,−1)
            then  if I am ⋆
                  then  Move to Position (1,3) //Fig. 14, Moving #3'
            else  if I am ○
                  then  1stSpiral(1) //Fig. 14, Moving #2
      else  if R_T* is an (1,3)-rectangle
            then  if ∘ = (−1,−1)
                  then  if I am ⋆
                        then  Move to Position (2,3) //Fig. 14, Moving #3"
                  else  if ∘ = (−2,−2)
                        then  if I am ○
                              then  Move to Position (−2,−1) //Fig. 14, Moving #6
                        else  // ∘ = (−2,−1)
                              if I am ⋆
                              then  2ndSpiralMngmt // Robot ⋆ executes the last spiral phase
            else  if R_T* is an (2,3)-rectangle
                  then  if ∘ = (−2,−2)
                        then  if I am ⋆
                              then  Move to Position (1,3) //Fig. 14, Moving #5
                        else  if I am ○
                              then  1stSpiral(2) //Fig. 14, Moving #4
```

Algorithm 10 Procedure $1stSpiral$.

```
Procedure 1stSpiral(phase)
    Let (x, y) be my current coordinates over S*
    if y = −phase
    then  if x = phase
          then  Move to Position (x, y + 1)
          else  if x ≠ −phase
                then  Move to Position (x + 1, y)
    else  if x = phase
          then  if y = phase
                then  Move to Position (x − 1, y)
                else  Move to Position (x, y + 1)
          else  if y = phase
                then  if x = −phase
                      then  Move to Position (x, y − 1)
                      else  Move to Position (x − 1, y)
                else
                      Move to Position (x, y − 1)
```

Algorithm 11 Procedure $2ndSpiralMngmt$ executed by Robot $\star$.

```
Let S° be the coordinate system built over R_To // d_To = 3
Let (x, y) be my current coordinates over S°
Let phase = max{|x|, |y|}
if phase < ⌊ℓ/2⌋
then  2ndSpiral(phase)
else  if phase = ⌊ℓ/2⌋
      then  if ℓ is odd
            then  if ℓ = L
                  then  if (x, y) = (−phase, −phase)
                        then  Exploration Done!
                        else  2ndSpiral(phase)
                  else  if (x, y) = (−phase, −phase)
                        then  Move to Position (x, y − 1)
                        else  Move to Position (x − 1, y)
            else  // Case ℓ is even
                  if ℓ = L
                  then  if (x, y) = (phase − 1, phase)
                        then  Exploration Done!
                        else  2ndSpiral(phase)
                  else  if (x, y) = (phase − 1, phase)
                        then  Move to Position (x, y + 1)
                        else  Move to Position (x + 1, y)
      else  // Case phase > ⌊ℓ/2⌋, i.e., L > ℓ
            if ℓ is odd
            then  Odd_Rectangle(phase)
            else  Even_Rectangle(phase)
```

Algorithm 12 Procedure $2ndSpiral$.

```
Procedure 2ndSpiral(phase)
    Let (x, y) be my current coordinates over S° // over T∘
    if y = −phase
    then  Move to Position (x − 1, y)
    else  if x = −phase
          then  if y = phase
                then  Move to Position (x + 1, y)
                else  Move to Position (x + 1, y)
          else  if y = phase
                then  if x = phase
                      then  Move to Position (x, y − 1)
                      else  Move to Position (x + 1, y)
                else  // x = phase
                      Move to Position (x, y − 1)
```

Algorithm 13 Procedure Odd_Rectangle.

```
Procedure Odd_Rectangle(phase)
    if phase + ⌈ℓ/2⌉ = L
    then  if |x| = phase
          then  Exploration Done!
          else  if phase is even
                then  Move to Position (x + 1, y)
                else  Move to Position (x − 1, y)
    else  if |x| ≠ phase
          then  Move to Position (x, y + 1)
          else  if phase is even
                then  Move to Position (x + 1, y)
                else  Move to Position (x − 1, y)
```

Algorithm 14 Procedure Even_Rectangle.

```
Procedure Even_Rectangle(phase)
    if phase + ⌊ℓ/2⌋ = L
    then  if |x| = phase
          then  Exploration Done!
          else  if phase is even
                then  Move to Position (x − 1, y)
                else  Move to Position (x + 1, y)
    else  if |x| ≠ phase
          then  Move to Position (x, y + 1)
          else  if phase is even
                then  Move to Position (x − 1, y)
                else  Move to Position (x − 1, y)
```

5 Correctness Proof

Recall that Algorithm 1 is made of three phases: SetUp, Tower, and Exploration. The first phase, SetUp, starts from any towerless configuration and ends in a $\Diamond$.Configuration. Moreover, this phase contains no configuration with a tower. The second phase, Tower, just consists in a single probabilistic transition (that may be performed in several steps) from a $\Diamond$.Configuration to the initial towered configuration of the Exploration phase. Finally, all configurations that appear during the Exploration phase contain a tower. Hence, we have the following property:

Remark 3 *The three phases of Algorithm 1 (SetUp, Tower, and Exploration) are unambiguous.*

Consider any $\Diamond$.Configuration. Let u_1 and u_2 be the two occupied neighboring nodes having themselves two occupied neighboring nodes. Let r_1 and r_2 be the two robots located at u_1 and u_2, respectively. During this phase, r_1 and r_2 *try to move* towards each other anytime they are activated; the other two robots do not move if activated. If either r_1 or r_2, but not both, is activated, then a tower is created. If both r_1 and r_2 are activated simultaneously, then the following two events have positive probability to occur: either the system remains a $\Diamond$.Configuration (because r_1 and r_2 both decide to either move or not move) or a tower is created (because either r_1 or r_2 decides to move). As the scheduler is fair, r_1 and r_2 are activated regularly until a tower is created, and with probability 1, the initial towered configuration of the Exploration phase is eventually reached.

Then, from the initial configuration of the Exploration phase, robots $\star$ and $\circ$ explore the torus in a fully deterministic manner. So, from such a configuration, the termination is certain. Thus, we can conclude that starting from a $\Diamond$.Configuration, the torus is eventually explored with probability one.

Hence, the critical part of the algorithm is the SetUp phase, whose aims at converging to a $\Diamond$.Configuration starting from any initial (towerless) configuration. So, we now focus on this phase and show that, the convergence is obtained with probability one.

First, the SetUp phase exactly deals with the set of all towerless configurations. Now, by definition, we have:

Remark 4 *The set of all towerless configurations is equal to the (disjoint) union of the following configuration sets:*

1. *$\Diamond$.Configurations,*
2. *Regular,*
3. *Double-Trap1,*
4. *Double-Trap2,*
5. *Twin,*
6. *Triplet,*
7. *Quadruplet, and*
8. *Isolated.*

The first part of the proof consists in showing that if the system is in a configuration of type Double-Trap1, Double-Trap2 or Triplet, then it (deterministically) reaches a $\Diamond$.Configuration in finite time.

In the following, we consider an arbitrary towerless configuration C.

Lemma 3 *If C is of type Double-Trap2, then a configuration C' of type $\Diamond$.Configuration is reached from C within finite time.*

Proof. When the configuration C is of type Double-Trap2, there is exactly one robot r that is allowed to move (refer to Algorithm 2, Lines 2.08-2.10). r is the robot that has a neighboring robot and another robot at distance 2 on the same ring. Its destination is the adjacent free node towards the robot that is at distance 2. As the scheduler is fair, r moves in finite time and then a configuration of type $\Diamond$.Configuration is reached. □

Lemma 4 *If C is of type Double-Trap1, then a configuration C' of type $\Diamond$.Configuration is reached from C within finite time.*

Proof. When the configuration C is of type Double-Trap1, the only robot allowed to move is the robot r that is at the extremity of the 3-block having exactly one neighboring robot (refer to Algorithm 2, Lines 2.04-2.07). Its destination is the adjacent free node on the ring containing a single robot. As the scheduler is

fair, r moves in finite time and a configuration of type Double-Trap2 is then reached. Hence, by Lemma 3, we are done. □

Lemma 5 *If C is of type Triplet, then a configuration C' of type $\Diamond$.Configuration is reached from C within finite time.*

Proof. Let R be the ring that contains three robots in C. The cases below are possible:

1. *R contains a 3-block.* In this case the robot that is not part of R is the only one allowed to move (refer to Algorithm 3, Lines 3.13-3.15). Its destination is the adjacent node on the shortest path towards the closest free node neighbor of one extremity of the 3-block (let us refer to this free node by U). Observe that this robot is the only one allowed to move unless it reaches U. Now, the scheduler being fair, the target node is reached in finite time and then a Double-Trap1 configuration is created. By Lemma 4, we are done in this case.
2. *R contains a 2-block.* In this case, the robot that is on R, but not part of the 2-block is the only one allowed to move (refer to Algorithm 3, Lines 3.10-3.09). Its destination is the adjacent free node on the shortest path towards the 2-block (on R). Note that this robot is the only one allowed to move unless it becomes part of a 3-block. Now, as the scheduler is fair, we retrieve Case 1 in finite time and we are done.
3. *The other cases.* (refer to Algorithm 3, Lines 3.03-3.09)
   (a) If $R \cap$ wall= Occupied-Node. Then, the robots allowed to move are the two robots on R that are not located at Occupied-Node. Their destination is the adjacent free node towards the wall. According to the choices of the fair scheduler, within finite time either a 3-block is created and we retrieve Case 1, or a 2-block is created and we retrieve Case 2.
   (b) If $R \cap$ wall= Free-Node then, every robot of R that is not neighbor of Free-Node moves towards Free-Node until a 2-block is created on R. As the scheduler is fair, a 2-block is created on R in finite time, and we retrieve Case 2.

□

From the other towerless configurations (Regular, Twin, Quadruplet, and Isolated), the convergence to a $\Diamond$.Configuration is *probabilistic*. Now, when considering any arbitrary fair execution, (1) the topology is fixed with, in particular, a *finite* number of nodes and (2) recall that the number of robots is the *constant 4*. So, the number of towerless configurations is also *finite*. Now, while a configuration of type Double-Trap1, Double-Trap2, Triplet, or $\Diamond$.Configuration is not reached, the SetUp phase remains trapped in the finite subset of towerless configurations defined as the distinct union of Configuration Regular, Twin, Quadruplet, and Isolated, by Remark 4. So, in order to demonstrate that the system eventually reaches a $\Diamond$.Configuration with probability one, we show below that from any configuration of type Regular, Twin, Quadruplet, or Isolated, there always exists a positive probability to reach a configuration of type Double-Trap1, Double-Trap2, Triplet, or $\Diamond$ in finite time and despite the choices of the fair scheduler. (Remember that once a configuration of type Double-Trap1, Double-Trap2, or Triplet is reached, a $\Diamond$.Configuration is reached within finite time, by Lemmas 3-5.)

Lemma 6 *If C is of type Twin then, with positive probability, a configuration of type Triplet is reached in finite time.*

Proof. When the configuration C is of type Twin, the cases below are possible (C cannot contains four couples, otherwise it would be of type Regular):

1. *C contains a unique couple, R.*
   (a) If there is a unique robot r that is outside R and the closest to R, then r is the only robot allowed to move, and if activated, it moves to an adjacent free node on a shortest path to a free node of

R (refer to Algorithm 4, Lines 4.02-4.04). Observe that by moving r remains the closest robot from R. Thus, r keeps being the only robot allowed to move until it becomes part of R. Hence, as the scheduler is fair, a Triplet configuration is reached, in this case, *within finite time*.

(b) In either cases, the two robots that are outside R. Let r be any of those robots. Let R_{per} be the ring of r perpendicular to R. Let R_{para} be the ring parallel to R which contains the other robot outside R. If activated, r *try to move* to a neighboring free node on R_{per} which does not belong to R or R_{para} (refer to Algorithm 4, Lines 4.05-4.08). Then, as the scheduler is fair, at least one of these two robots is activated in finite time, and when activated, with positive probability (actually, a probability greater or equal to $p(1-p)$), exactly one of those robots moves and we retrieve the previous case. Hence, we can conclude that, in this case, *with positive probability*, a configuration of type Triplet is reached *in finite time*.

2. *C contains two couples.* In this case, some of the robots (at least one) have a neighboring safe node outside any couple. Only these robots are allowed to move (Algorithm 4, Lines 4.10-4.12). If activated, they try to move to a neighboring safe node outside any couple. As the scheduler is fair, one of them is activated within finite time and when activated, with positive probability (actually, a probability greater or equal to $p(1-p)^3$), exactly one of them moves. In this case, the system reaches a configuration Twin containing only one couple (Case 1). Hence, *with positive probability*, a configuration of type Triplet is reached *in finite time*.
3. *C contains three couples.* (In this case, the torus necessarily satisfies $\ell = L$.) Observe that, in this case, there exist two robots that belong to two couples. Only those robots are allowed to move. If activated, they *try to move* to a neighboring safe node (refer to Algorithm 4, Lines 4.13-4.15). As the scheduler is fair, at least one of them is activated within finite time and when activated, with positive probability (actually, a probability greater or equal to $p(1-p)$), exactly one of them moves and the configuration reached is still Twin but with two couples, that is, one of the previous cases. Hence, we can conclude that, in this case, *with positive probability*, a configuration of type Triplet is reached *in finite time*.

□

Lemma 7 *If C is of type Quadruplet then, a configuration C' of type either Twin or Triplet is reached from C within finite time.*

Proof. Let us refer to the ring that contains four robots in C by R. We consider the subgraph G_R induced by the nodes of the ring R that contains the 4 robots. We have the following possible cases:

- If *G_R contains exactly one smallest hole.* In this case, the two robots neighbor to such a hole move to an adjacent free node outside R, if activated (refer to Algorithm 5, Lines 5.03-5.05). As the scheduler is fair, at least one of them is activated *within finite time*. Then, if both robots move simultaneously, then the reached configuration is of type Twin. If only one robot moves, then the reached configuration is of type Triplet.
- If *G_R contains exactly one biggest hole.* In this case too, the two robots neighbor to such a hole move to an adjacent free node outside R, if activated (refer to Algorithm 5, Lines 5.06-5.08). As the scheduler is fair, at least one of them is activated *within finite time*. Then, if both robots move simultaneously, the reached configuration is of type Twin. If only one robot moves, then a Triplet configuration is reached.
- Otherwise, there are at most two robots that are neighbor to the two smallest holes on G_R. These robots are the only ones allowed to move. Their destination is the adjacent free node outside R (refer to Algorithm 5, Lines 5.09-5.10). As the scheduler is fair, one or two of those robots move in *finite time* and then a configuration of type Triplet or Twin is reached.

From the cases above we can deduce that starting from C, either Twin or Triplet configuration is reached in finite time. □

By Lemma 6, we have the following corollary from the previous lemma:

Corollary 2 *If C is of type Quadruplet then, with positive probability, a configuration of type Triplet is reached in finite time.*

Lemma 8 *If C is of type Isolated then, with positive probability, a configuration of type Triplet is reached in finite time.*

Proof. We split the study into two main cases:

1. $\ell = L$. There are two cases:
   - *There is a unique smallest enclosing rectangle s.* We call *internal robot* any robot that is not located on any side of s. The following subcases are possible:
     (a) *There is exactly one robot that is at a corner c of s.* In this case there is exactly one internal robot. It is the only robot allowed to move, its destination is the closest free node on a side of s having c as extremity (refer to Algorithm 6, Lines 6.07-6.09). Observe that this robot keep being enabled until it becomes on the same ring as c (the SER is invariant). Thus, as the scheduler is fair, a Twin configuration is reached in *finite time* and by Lemma 6, we are done.
     (b) *There are exactly two robots at two corners c_1, c_2 of s.* The other two robots are in this case internal (otherwise the configuration would contain a couple). The internal robots that are the closest to an occupied corner of s move towards the closest free node on a side of s having c_1 or c_2 as an extremity (refer to Algorithm 6, Lines 6.10-6.12). Note that in the case where there are two robots allowed to move, their target destination is different (recall that the configuration is of type Isolated). The robots that have moved keep moving towards the same target node until they become on the same ring as a robot on the corner of s. Thus, as the scheduler is fair, a Twin configuration is reached in *finite time* and by Lemma 6, we are done.
     (c) *There is no robot at the corner of s.* In this case, observe that all robots are located at different sides of s. Let d be the minimum distance between any occupied node and a corner of s. The following subcases are then possible:
       i. *There is a unique robot r that is at distance d from a corner of s.* r is the only robot allowed to move. Its destination is the adjacent free node towards the closest corner (refer to Algorithm 6, Lines 6.15-6.17). Observe that by moving r becomes even close to the corner, thus r keeps being the only one enabled to move until it becomes neighbor of the closest corner. Hence, as the scheduler is fair, we retrieve Case 1a within *finite time*, and we are done.
       ii. *There are two robots r_1 and r_2 that are at distance d from a corner.*
         A. If r_1 and r_2 are closest to the same corner c, then the two other robots, say r_3 and r_4, move towards the corner that is common to their side of s and the side of s where either r_1 or r_2 are located, if activated (refer to Algorithm 6, Lines 6.18-6.21). Note that since all robot are on the border of s, r_3 and r_4 have different targets. Thus, as the scheduler is fair, a Twin configuration is reached in *finite time* and by Lemma 6, we are done.
         B. If r_1 and r_2 are closest to a corner of s, but not the same then, then r_1 and r_2 are the only robots allowed to move. Their destination is the adjacent free node towards the closest corner (refer to Algorithm 6, Lines 6.22-6.23). As the scheduler

is fair, in *finite time*, either they both reach their target simultaneously and we retrieve Case 1b, or we retrieve Case 1a, Case 1.(c).i, or the system reaches a configuration of type Twin. In the latter case, we can conclude with Lemma 6. In all other cases, we retrieve of the previous cases, and we are done.

iii. *There are three or four robots that are at distance d from a corner of s.* Observe that since that the configuration is not of type Regular, there is one or two corners that have only one robot at distance d from them. Such a robot is enabled to move and its destination is the closest corner (refer to Algorithm 6, Lines 6.22-6.23). As the scheduler is fair, in *finite time*, we retrieve either Case 1a or Case 1b and we are done.

- *There are several possible smallest enclosing rectangles.* In this case, some of the robots (at least one) have a safe neighboring node such that if they move to that node and they move alone, then the number of SER decreases. Only these robots are allowed to move. If activated, they try to move to such a safe node (Algorithm 6, Lines 6.01-6.03). As the scheduler is fair, at least one of them is activated within finite time and when activated, with positive probability (actually, a probability greater or equal to $p(1-p)^3$), exactly one of them moves. In this case, the system reaches configuration Twin and we are done (Lemma 6), or the system is still in a configuration Isolated, yet the number of possible smallest enclosing rectangles has strictly decreased.
  Hence, in the worst case, the system requires three consecutive sequential moves (which happens with a probability greater or equal to $p^3(1-p)^9$) to reach either a Twin configuration or a configuration Isolated containing a unique smallest enclosing rectangle. Hence, with positive probability the system reaches in *finite time* a Twin configuration and by Lemma 6, we are done.

2. $\ell \neq L$. In this case, according to the number of robots on L-rings, we can distinguish the following cases:
   (a) *Every L-ring contains at most one robot.* Let r_1, r_2, r_3, and r_4 be the four robots.
   Assume there is a unique smallest enclosing rectangle s. Observe that there are two robots, say r_1 and r_2, that are on two parallel sides of s that are on ℓ-rings. The robots allowed to move are r_3 and r_4. If activated, they move along their L-ring toward the closest ℓ-ring which is a side of s (refer to Algorithm 7, Lines 7.01-7.10). Note that by doing so, r_3 and r_4 keeps being allowed to move until they become on the same ℓ-ring as either r_1 or r_2. Hence, as the scheduler is fair, Twin configuration is reached in *finite time* and by Lemma 6, we are done.
   If there are several smallest enclosing rectangles, we proceed as in the case $\ell = L$: In the worst case, the system requires three consecutive sequential moves (which happens with a probability greater or equal to $p^3(1-p)^9$) to reach either a Twin configuration or a configuration Isolated containing a unique smallest enclosing rectangle. Hence, with positive probability the system reaches in *finite time* a Twin configuration and by Lemma 6, we are done.
   (b) *There is exactly one L-ring R that contains 2 robots.* The robots that are alone on their L-ring move along their L-ring towards a closest free node having a node of R in its ℓ-ring (refer to Algorithm 7, Lines 7.11-7.13). Observe that these robots keep the same target until one of them becomes on the same ℓ-ring as another robot. Then, in *finite time*, the system reaches either a Triplet or a Twin configuration depending of the activations decided by the fair scheduler. In the latter case, with a *positive probability*, a Triplet configuration is then reached in *finite time* by Lemma 6 and we are done.
   (c) *There is exactly one L-ring R that contains 3 robots.* The case is similar to the previous one, refer to Algorithm 7, Lines 7.11-7.13: in finite time, the system reaches a Twin configuration and by Lemma 6, we are done.
   (d) *There exist two L-rings that contain two robots.* In this case, some of the robots (at least one) have a safe neighboring node outside their current L-ring. Only these robots are allowed to move (Algorithm 7, Lines 7.14-7.16). If activated, they try to move to safe neighboring node outside

their current L-ring. As the scheduler is fair, one of them is activated within finite time and when activated, with positive probability (actually, a probability greater or equal to $p(1-p)^3$), exactly one of them moves. In this case, we retrieve cases 2b or 2c, and we are done.

(e) *There is one L-ring R that contains all robots.* Let G_R be the subgraph induced by R. The robots in this case apply Procedure Quadruplet on G_R. Similarly to the proof of Lemma 7, one or two robots leave R within finite time, and we retrieve one of the previous cases.

□

Lemma 9 *If C is of type Regular then, with positive probability, a non-Regular towerless configuration is reached in finite time.*

Proof. If C is of type Regular then every robot *tries to move* to a safe node, if activated. Now, in this case, with positive probability, exactly one of them moves during the next step and a non-Regular towerless configuration is then reached (more precisely, a configuration of type $\Diamond$, Isolated, Twin, Triplet, or Double-Trap2). □

From all previous lemmas and corollary, we can deduce that, with probability one, the SetUp phase eventually terminates in a $\Diamond$.Configuration. Hence, we can conclude with the following theorem:

Theorem 4 *Assuming any fair scheduler, Algorithm 1 is a probabilistic exploration protocol for 4 robots on any torus of size $\ell \times L$, where $7 \leq \ell \leq L$.*

6 Concluding Remarks

While the solution we provided for the torus exploration problem is optimal in terms of number of robots, there remain challenging open questions. First, we presented an algorithm for all tori of size $\ell \times L$, where $7 \leq \ell \leq L$. In [3], the authors stated that small grids require more robots. Determining if our results can be extended to smaller tori is an interesting problem. We expect mechanized approaches [17] to be valuable for investigating small size tori. Second, dealing with higher dimension (*e.g.*, from a ring to a torus) does not necessarily increase the robot number complexity of the exploration problem. The issue of the d-dimensional tori (with $d > 2$) remains open.

References

[1] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.

[2] J. Chalopin, P. Flocchini, B. Mans, and N. Santoro. Network exploration by silent and oblivious robots. In *WG*, pages 208–219, 2010.

[3] Stéphane Devismes, Anissa Lamani, Franck Petit, Pascal Raymond, and Sébastien Tixeuil. Optimal grid exploration by asynchronous oblivious robots. In *SSS*, pages 64–76, 2012.

[4] Stéphane Devismes, Franck Petit, and Sébastien Tixeuil. Optimal probabilistic ring exploration by semi-synchronous oblivious robots. *Theor. Comput. Sci.*, 498:10–27, 2013.

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

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

[7] Roberto Baldoni, François Bonnet, Alessia Milani, and Michel Raynal. Anonymous graph exploration without collision by mobile robots. *Inf. Process. Lett.*, 109(2):98–103, 2008.

[8] Roberto Baldoni, François Bonnet, Alessia Milani, and Michel Raynal. On the solvability of anonymous partial grids exploration by mobile robots. In *OPODIS*, pages 428–445, 2008.

[9] Gianlorenzo D'Angelo, Gabriele Di Stefano, Alfredo Navarra, Nicolas Nisse, and Karol Suchan. A unified approach for different tasks on rings in robot-based computing systems. In *IPDPS Workshops*, pages 667–676, 2013.

[10] Gianlorenzo D'Angelo, Alfredo Navarra, and Nicolas Nisse. Gathering and exclusive searching on rings under minimal assumptions. In *ICDCN*, pages 149–164, 2014.

[11] K. Haba, T. Izumi, Y. Katayama, N. Inuzuka, and K. Wada. On gathering problem in a ring for 2n autonomous mobile robots. In *SSS*, page Poster, 2008.

[12] Ralf Klasing, Adrian Kosowski, and Alfredo Navarra. Taking advantage of symmetries: Gathering of many asynchronous oblivious robots on a ring. *Theor. Comput. Sci.*, 411(34-36):3235–3246, 2010.

[13] Anissa Lamani, Maria Potop-Butucaru, and Sébastien Tixeuil. Optimal deterministic ring exploration with oblivious asynchronous robots. In *SIROCCO*, pages 183–196, 2010.

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

[15] Daniel W. Stroock. *Probability theory : an analytic view*. Cambridge university press, Cambridge, New York, 1993.

[16] Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. In *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, Lecture Notes in Computer Science (LNCS), pages 105–118. Springer Berlin / Heidelberg, December 2007.

[17] François Bonnet, Xavier Défago, Franck Petit, Maria Potop-Butucaru, and Sébastien Tixeuil. Discovering and assessing fine-grained metrics in robot networks protocols. In *33rd IEEE SRDS Workshops, Workshop on Self-organization in Swarm of Robots*, pages 50–59, 2014.