Tight Bounds for Asynchronous Collaborative Grid Exploration

Sebastian Brandt\textsuperscript{1}, Jara Uitto\textsuperscript{*1,2}, and Roger Wattenhofer\textsuperscript{1}

\textsuperscript{1}ETH Zürich
\textsuperscript{2}University of Freiburg

Abstract
Consider a small group of mobile agents whose goal is to locate a certain cell in a two-dimensional infinite grid. The agents operate in an asynchronous environment, where in each discrete time step, an arbitrary subset of the agents execute one atomic look-compute-move cycle. The protocol controlling each agent is determined by a (possibly distinct) finite automaton. The only means of communication is to sense the states of the agents sharing the same grid cell. Whenever an agent moves, the destination cell of the movement is chosen by the agent’s automaton from the set of neighboring grid cells. We study the minimum number of agents required to locate the target cell within finite time and our main result states a tight lower bound for agents endowed with a global compass. Furthermore, we show that the lack of such a compass makes the problem strictly more difficult and present tight upper and lower bounds for this case.

1 Introduction
In this paper, we consider the problem of exploring an infinite grid with a set of mobile agents controlled by asynchronous finite state machines. Our goal is to find the minimum number of agents required to perform this task in finite time. We consider two different models, where the first model assumes a globally consistent orientation, i.e., every agent is endowed with a compass. In this model, a relatively simple algorithm is known to this problem that requires four agents and a lower bound of three agents can be derived with a moderate amount of work, it turns out that adding another agent into the picture adds a considerable amount of complexity. The main result of this paper is to show a lower bound of four agents for this variant of the problem.

The models considered in this paper is motivated by modeling entities with limited capabilities, such as ants or low cost mobile robots. A single ant is limited in its capabilities, yet a swarm of ants seems to be more than the sum of its parts. A remarkable feature of ants is that they manage to find food sources in seemingly arbitrary locations. This food gathering problem is easy to

\textsuperscript{*}This work was partially supported by ERC Grant No. 336495 (ACDC).
describe mathematically, and as such is the problem of choice when it comes to theoretically understanding the limits of distributed mobile agents.

A finite state machine seems to be a good theoretical model to express limited capabilities. As ants do not perform a perfectly timed dance, the asynchronous model must be applied. An asynchronous finite state machine is clearly limited; it is easy to see that a single ant can only find a food source by sheer luck. Furthermore, it is reasonable to assume that ants are able to exchange a limited amount of information with physical contact.

We also consider a variant of the model from previous work, in which we remove the assumption of a global orientation. Our attempt is to model weak mobile agents by allowing means to backtrack a few steps through a port numbering, while disallowing long consistent movements easily obtained with a compass. We show a lower bound of five asynchronous agents in this model and match it with a protocol for five agents that explores the grid in finite time.

2 Related Work

Graph exploration is a widely studied problem in the computer science literature. In the typical setting one or more agents are placed on some node of a graph and the goal is to visit every node and/or edge of the graph by moving along the edges. There is a wide selection of variants of graph exploration and one of the standard ways to classify these variants is to divide them to directed and undirected variants [11, 1]. In the directed model, the edges of the graph only allow traversing into one direction, whereas in the undirected model, traversing both ways is allowed. Our work assumes the undirected graph exploration model.

Other typical parameters of the problem are the conditions of a successful exploration and symmetry breaking mechanisms. Some related works demand that the agents are required to halt after a successful exploration [12] or that the agents must return to their starting point after the exploration [3]. We only require that one agent enters a specific adversarially selected cell, but we note that the halting property is trivial to obtain and the origin can be relocated by essentially reverting our exploration protocol. From the perspective of symmetry breaking, one characterization is to break the problem into the case of equipping nodes with unique identifiers [23, 15] and into the case where nodes are anonymous [8, 24, 5]. Since the memory of our agents is restricted to a constant amount of bits with respect to the size of the graph, the unique identifiers are not helpful.

We consider the port numbering model, where the agents are locally able to distinguish between the edges leading to the neighboring nodes. This can be thought of as labeling the incident edges of node \( u \) with labels \( 1, \ldots, d(u) \), where \( d(u) \) denotes the degree of \( u \). The port numbering model can be further divided into cases, where on one extreme, the port numbering is globally consistent, e.g., in the case of a grid all nodes and for every agent, the ports 1, 2, 3 and 4 lead to north, west, east, and south, respectively. This can be thought of a compass that always points out the cardinal directions. On the other extreme, the port numbering is restricted to a Local Orientation [14], i.e., there is no consistency between the edge labels. In our model, an agent is aware through which port it arrived to its current node. Our main result assumes the globally consistent model and we use the same ideas to show a stronger lower bound in a port
numbering model without such a global labeling.

The agents typically operate in look-compute-move cycles, where they first gather the local information, then perform local computations, and finally, decide to which node they move. In our case, this corresponds to sensing the states of the agents in the same cell and applying the transition function. This execution model can be divided into synchronous [26], semi-synchronous [25] and asynchronous variants [27, 19], referred to as FSYNC, SSYNC, and ASYNC. In the FSYNC model, all agents execute their cycles simultaneously in discrete rounds. In SSYNC model only a subset (not necessarily proper) of the agents is activated in every round and in the ASYNC model, even the number of cycles per discrete round is not bounded. To avoid confusion, we refer to the non-synchronous rounds as time steps. In this paper, we consider the semi-synchronous model. Note that since the ASYNC model is weaker than the SSYNC model, we directly obtain our lower bound result for this model as well.

The standard efficiency measure of a graph exploration algorithm executed in the FSYNC model is the number of synchronous rounds it takes until the graph is explored [23]. In the non-synchronous models, this measure is typically generalized to the maximum delay between activation times of any agent [9]. A widely-studied classic is the cow-path problem, where the goal of the cow is to find food or a treasure on a line as fast as possible. There is an algorithm with a constant competitive ratio for the case of a line and in the case of a grid, a simple spiral search is optimal and the problem has been generalized to the case of many cows [4, 22]. Some more recent work studied the time complexity of \( n \) distributed agents searching for a treasure in distance \( D \) on a grid and a \( \Theta(D/n^2 + D) \) bound was shown in the case of Turing machines without communication and in the case of communicating finite state machines [18, 17].

Our work does not focus on the time complexity of the problem, but rather on the computability, i.e, what is the minimum number of agents that are required to find the treasure. The canonical algorithm in the case of little memory is the random walk, where the classic result states that a random walk explores an \( n \)-node graph in polynomial time [2]. In the case of infinite grids, it was shown in a recent paper that, even with a globally consistent orientation, two randomized agents cannot locate the treasure in finite expected time [10]. Combining with the results from [16], it follows that this lower bound is tight. In the deterministic case, our lower bound of four deterministic asynchronous agents closes the remaining gap in the results of [16].

Another typical measure for efficiency is the number of bits of memory needed per agent [12, 20]. For example, it was shown by Fraigniaud et. al., that \( \Theta(D \log \Delta) \) bits are needed for a single agent to located the treasure, where \( D \) and \( \Delta \) denote the diameter and the maximum degree of the graph, respectively. The memory of our agents is bounded by a universal constant, independent of any graph parameters. Work that falls somewhat close to our work is the study of graph exploration in labyrinths, i.e., graphs that can be embedded to a 2-dimensional grid and some subset of nodes cannot be entered by the agents. The classic results state that all co-finite (finite amount of cells not blocked) labyrinths can be explored by two finite automata and an automaton with two pebbles [7], and that finite labyrinths (finite amount of cells are blocked) can be explored using one agent with four pebbles [5], where a pebble is a movable marker. Furthermore, it is known since long that there are finite and co-finite labyrinths where one pebble is not enough [21] and that there are graphs where
no finite number of finite automata suffices [24]. More recently, it was shown that $\Theta(\log \log n)$ pebbles for an agent with $\Theta(\log \log n)$ memory is the right answer for general graphs [13].

3 Model

Consider $n$ mobile agents that explore $\mathbb{Z}^2$. Initially, all agents are positioned in the same grid cell referred to as the origin (say, the cell with coordinates $(0, 0) \in \mathbb{Z}^2$). The agents cannot distinguish between the origin and the other cells.

We define the distance $\text{Dist}(c, c')$ according to the Manhattan distance between two grid cells $c = (x, y)$ and $c' = (x', y')$ in $\mathbb{Z}^2$, i.e., $|x - x'| + |y - y'|$. We also denote the distance between agent $a$ and cell $c$ and the distance between agents $a$ and $a'$, by $\text{Dist}(a, c)$ and $\text{Dist}(a, a')$, respectively. Two cells are called neighbors if the distance between them is 1. In each step of the execution, agent $a$ positioned in cell $(x, y) \in \mathbb{Z}^2$ can either move to one of the four neighboring cells $(x, y + 1), (x, y - 1), (x + 1, y), (x - 1, y)$, or stay put in cell $(x, y)$. We consider two variants of the model with respect to the sense of orientation. The first variant considers agents that are equipped with a compass, that is, the cardinal directions $(N, W, S, E)$ are aligned for every agent in every cell. In the second variant, we remove the assumption of having a compass. Instead, every grid cell has a port numbering, i.e., upon deciding on a move, every agent can choose between ports labeled with $(1, 2, 3, 4)$ that lead to distinct neighboring cells. In the terminology of port numbers, the compass can be identified with a labeling that is consistent throughout the grid, e.g., port 1 always points north and so on. We assume neither that the port numbers between two adjacent cells are consistent nor that the port numbering is consistent between two different agents, i.e., port $i$ from cell $c$ can lead agent $a$ to cell $c'$ and agent $a'$ to cell $c''$, where $c' \neq c''$.

We provide the agents with the information of through which port they can return to the previously occupied cell. To ensure that the agents can navigate back according to a (finite) sequence of ports that they stored in their memory, we assume that the port numbering always remains consistent in an $O(1)$-hop neighborhood of each agent, where the constant in the $O$ notation is bounded by the size of the automaton. To help us fight through some technical details in the model without a compass, we allow the adversary to dynamically change the port numbers that are further than constant hops away from the agent. We say that agent $a$ moves according to a port sequence, i.e., an array of numbers between 1 and 4, when the movements of $a$ follow the numbers in the sequence.

The protocol of each agent only has a finite set of states, i.e., every agent is controlled by a finite automaton. Since we only consider instances with a constant number of agents, we allow each agent to run a different individual protocol. This is modeled by assigning to each agent an individual initial state in their mutual automaton.

The communication and computational capabilities of the agents are as follows. An agent $a$ positioned in cell $c \in \mathbb{Z}^2$ can communicate with all other agents positioned in cell $c$ at the same time. Agent $a$ senses for each state of the finite automaton of any agent, whether there exists an agent $a' \neq a$ in cell $c$ whose current state is $q$. Formally, the protocol $\Pi$ of an agent $a$ is a three-tuple
\( \langle Q, s^0, \delta \rangle \), where \( Q \) is the set of states of the finite automaton, \( s^0 \) is the initial state of agent \( a \) and \( \delta = Q \times \{0, 1, 2, 3, 4\} \times 2^Q \rightarrow 2^Q \times \{0, 1, 2, 3, 4\} \) is the transition function. Note that \( \delta \) takes the current state of an agent, the states it senses, and the port that it entered the current cell as its arguments and outputs a new state combined with a movement. Not performing a movement is identified with choosing port 0, which is equivalent to assigning a self loop to every cell. Note that the finite memory of an agent implies that the agent can store a port sequence whose length is bounded by \( Q \).

The agents operate in an asynchronous environment, i.e., each agent’s execution progresses in discrete (asynchronous) steps indexed by the non-negative integers. In every step, an agent performs a look-compute-move cycle, i.e., first an agent senses the states of the other agents in the current cell. Then, the transition function is applied with the sensed states and the current state as an input to the transition function. Finally, the movement indicated by the transition function is performed. We denote the time at which agent \( a \) completes step \( i > 0 \) by \( t_a(i) > 0 \) and call \( t_a(i) \) an activation time. Following common practice, we assume that the activation times \( t_a(i) \) for all \( a \) and \( i \) are determined by the policy \( \psi \) of an adversary that is aware of the protocol. In order to prevent the adversary from delaying a single agent arbitrarily long, we require a policy to activate each agent at least once every time unit, i.e., for all agents \( t_a(1) \leq 1 \) and \( t_a(i + 1) - t_a(i) \leq 1 \) for \( i > 0 \). We call the set of activation times determined by the adversary a schedule. Therefore, the policy of an adversary maps the set of possible protocols to the set of possible schedules.

The goal of the agents is to find an adversarially hidden treasure. The treasure is hidden in some cell \( c \) in some distance \( d \) independent of the size of the finite automaton. Both of these parameters are unknown to the agents. A protocol is correct if it successfully locates the treasure within finite time, that is, brings at least one agent to cell \( c \) in finite amount of time units. An equivalent formulation of the problem is that the agents have to explore every cell of the grid. Accordingly, we often refer to the agents’ task as exploring the grid.

4 Preliminaries

For our lower bounds we will only use adversarial schedules where no two agents are scheduled at the same time. Any such schedule provides a total order on the actions of the agents. W.l.o.g., we assume that the configurations of the agents on the grid (including the information about the states they are currently in) occur at integer points in time \( t = 0, 1, \ldots \) and that the actions of the agents determining the transition from one configuration to a new one take place between these points in time where between each two subsequent points in time exactly one agent is scheduled according to the total ordering. If an agent’s action is scheduled between time \( t \) and \( t + 1 \), we say, for the sake of simplicity, that the action takes place at time \( t \). Each action of a scheduled agent consists of a move on the grid to an adjacent cell (where we allow null moves, i.e., staying in the currently occupied cell) and an update of the current state of the finite automaton of that agent.

We denote the cell an agent \( a \) occupies at time \( t \) by \( c_t(a) = (x_t(a), y_t(a)) \). Similarly, we denote the state of the finite automaton agent \( a \) is in at time \( t \) by \( q_t(a) \). If \( a = a_i \) for some \( 1 \leq i \leq 3 \), then we also write \( c_t^i, x_t^i, y_t^i, q_t^i \) instead
of \(c_t(a_i), x_t(a_i), y_t(a_i), q_t(a_i)\), respectively. Moreover, we denote the number of states of the finite automaton governing the behavior of the three agents by \(N\).

In our lower bound proofs, we show for each finite automaton that a certain number of agents governed by this automaton are not sufficient to explore the grid. In this context, we consider the number \(N\) as a constant, which also implies that the result of applying any fixed polynomial function to \(N\) is also a constant.

Let \(\ell\) be an infinite line in the Euclidean plane and \(D\) some positive real number. Let \(B\) be the set of all points in the plane with integer coordinates and Euclidian distance at most \(D\) to \(\ell\). Let \(B'\) be the set of all grid cells that have the same coordinates as some point in \(B\). Then we call \(B'\) a band.

**A Single Agent** Consider a single agent \(a\) moving on the grid. Since the number of states of its finite automaton is finite, \(a\) must repeat a state at some point, i.e., there must be points in time \(t, t'\) such that \(q_t(a) = q_{t'}(a)\) and \(q_{t''}(a) \neq q_t(a)\) for all \(t < t'' < t'\). As shown in [16], agent \(a\) will then, starting at time \(t'\), repeat the exact behavior it showed starting at time \(t\) regarding both movement on the grid and updating of its state. We call the 2-dimensional vector \(c_t(a) - c_{t'}(a) = (x_t(a) - x_{t'}(a), y_t(a) - y_{t'}(a))\) the travel vector of agent \(a\) (from time \(t\) to time \(t'\)). Moreover, we call the time difference \(t' - t\) the travel period.

In the case of multiple agents, we use the same definitions for any time segment where only a single agent is scheduled and does not encounter another agent. In particular, we can only speak of a travel vector and a travel period when there are two points in time (in the considered time segment) where the scheduled agent repeats a state and at both times as well as in the time between, the agent is alone in its cell.

**5 Three Asynchronous Agents Do Not Suffice**

In this section, we prove that three agents do not suffice to explore the grid, even when they are endowed with a compass, i.e., even when the agents know at all times which of the adjacent cells is to the north, east, south and west, respectively. The proof we present is a proof by contradiction. In the following we give a (very informal and inaccurate) high-level overview of how it proceeds.

**5.1 Proof Overview**

Our assumption, that holds throughout Section 5, is that three agents actually suffice to explore the grid. From this assumption, we derive a contradiction as follows: First, we fix an adversarial schedule for the three agents that has certain advantageous properties. (We will show that it is already possible to derive a contradiction for this specific schedule.) Then, using the finiteness of configurations of agents in any bounded area, we show that for each distance \(D\) there is a point in time such that from this time onwards, there are always at least two agents that have distance at least \(D\). However, since we can prove that any two agents must meet infinitely often, there must be infinitely many travels between the two far-away agents (which are not always the same agents). We show that the vector along which such a travel takes place must have a fixed slope that is the same for all such travel vectors (from a sufficiently large point in time on). Otherwise, there would be two subsequent travels forth and back.
of different slope which would imply that the traveling agent on its way back would miss the agent it is supposed to meet. This also holds if the traveling agent explores some area to the left and right of its travel direction, since the distance $D$ between the two endpoints can be made arbitrarily large.

The crucial part of the proof is to show that the state of the traveling agent at the end of its travel does not depend on the exact vector between the start and the endpoint of its travel, but only on this vector “modulo” some other vector $v$ that is obtained by combining all of the finitely many possible traveling vectors of the aforementioned fixed slope. This enables us to prove that, at the start of a travel, the information 1) about the states and relative locations “modulo $v$” of the agents, and 2) about which agent is scheduled next and which is the traveling agent, are sufficient to determine the same information at the start of the next travel. Since there are only finitely many of these information tuples (exactly because it contains only the modulo version of the relative locations), at some point a tuple has to occur again. Hence, in a sense, the whole configuration consisting of the three agents repeats its previous movement from this point on, not counting any movement in the direction of the fixed slope. Thus, in each repetition between two occurrences of the information tuple the whole configuration moves by some fixed (and always the same) vector, which implies that the agents explore “at most half” of the grid.

5.2 The Schedule

We now specify a schedule that we assume to be the adversarial schedule for the remainder of Section 5. We first schedule agent $a_1$ for some number of time steps, then agent $a_2$, then $a_3$, and then we iterate, again starting with $a_1$. The number of steps an agent is scheduled can vary. In other words, we can describe our schedule as a sequence $S = (S_1^1, S_2^1, S_3^1, S_1^2, S_2^2, S_3^2, S_1^3, \ldots)$ of subschedules where in each subschedule $S_i^j$ only agent $a_i$ is scheduled. The number of time steps in a subschedule $S_i^j$ is determined as follows:

1. If there is a (finite) number $u > 0$ of time steps after which agent $a_i$ is in a cell occupied by another agent, then the subschedule $S_i^j$ ends after $u_{\text{min}}$ time steps where $u_{\text{min}}$ denotes the smallest such $u$.

2. If Case 1 does not apply, but there is a (finite) number $u > 0$ of time steps after which $a_i$ is in the same state in the same cell as it was at some earlier point in time during $S_i^j$ (excluding the starting time of $S_i^j$), then do the following: Fix a total order on the state space of $a_i$’s finite automaton. (This total order can be chosen arbitrarily, but in each application of Case 2 for agent $a_i$ the same order has to be used.) Let $q$ be the smallest state according to this order which $a_i$ assumes at least twice in the same cell (if we scheduled $a_i$ indefinitely). Then $S_i^j$ ends at the first point in time at which $a_i$ is in state $q$ and in a cell where $a_i$ would assume $q$ at least twice. Note that the property that $a_i$ would assume $q$ twice implies that it would repeat the exact behavior between the first and the second assumption of $q$ infinitely often afterwards, thus iterating through the exact same movement on and on.

3. If none of the two above cases occurs, i.e., $a_i$ would move on indefinitely without meeting any other agent or being in the same state in the same
Figure 1: Figure 1a illustrates Case 2 of our schedule. In this case, an agent is scheduled until some state $q$ in some cell $c$ occurs for the second time. In Figure 1a, we illustrate Case 3 where one agent alone, walks infinitely far away from the other two agents. This case cannot occur, since otherwise, the two remaining agents would have to explore the grid without the third agent.

cell as before, then we schedule as follows: Let $(x, y)$ be the travel vector of $a_i$’s movement, and $k$ the travel period. Then the subschedule $S_{ij}^t$ ends at the first time $t$ (after the start of $S_{ij}^t$) for which the following property is satisfied: For each cell $(x_r^t, y_r^t)$ occupied by an agent $a_r, r \neq i$, we have that 1) $x_i^t - x_r^t > k$ if $x > 0$, and $x_i^t - x_r^t < -k$ if $x < 0$, and 2) $y_i^t - y_r^t > k$ if $y > 0$, and $y_i^t - y_r^t < -k$ if $y < 0$. The definition of the travel vector ensures that there is such a (finite) point in time $t$. Note that Case 3 can only occur if $x \neq 0$ or $y \neq 0$.

Moreover, if this case actually occurs, then the complete subsequent schedule is adapted according to the following special rule (overriding all of the above): After time $t$, the two agents $a_r, r \neq i$ are scheduled for one time step each (in arbitrary order), then agent $a_i$ is scheduled for $k$ time steps, i.e., exactly one travel period, and then we iterate this new scheduling.

Note that according to this schedule, the number of time steps a scheduled agent can stay put in a cell during one of its subschedules is upper bounded by $N$. We now collect a few lemmas that highlight certain properties of the three cases.

**Lemma 1.** Case 3 cannot occur.

*Proof.* Assume that Case 3 occurs and let $a_i$ denote the agent that would move on indefinitely without meeting another agent. Then, at the beginning of the first iteration according to the special rule, the distance of agent $a_i$ to any of the other agents is more than $k$ in at least one (of $x$- and $y$-) direction and $a_i$ moves away from the agents according to the travel vector. After each of the other agents makes a step, this distance is still at least $k$. Hence, agent $a_i$ cannot encounter one of the other agents during its next $k$ steps, since in total it moves away from the other agents, according to the specification of Case 3. The direction of the travel vector also ensures that the distance to the other agents is again increased to more than $k$ (in at least one direction). Thus, the same arguments hold for the next iteration, and we obtain by induction that agent $a_i$ will never encounter another agent after the occurrence of Case 3. It follows that, if three agents suffice to explore the grid, then also a team of two agents and a separate single agent can explore the grid without any communication between the team and the single agent. From [16], we know that this is not possible since two agents can only explore a band of constant width. 



8
Following Lemma 1 we will assume in the following that Case 3 does not occur, i.e., each agent’s subschedule ends because it encounters another agent or because it repeats a pair state/cell. This allows us to group the possible subschedules of an agent into two categories: We say that a subschedule \( S_j^i \) is of type 1 if \( S_j^i \) ends because of the condition given in Case 1 and of type 2 if \( S_j^i \) ends because of the condition given in Case 2.

**Lemma 2.** Any subschedule of type 2 consists of at most \( N \) time steps.

**Proof.** Assume for a contradiction that there is a subschedule \( S_j^i \) of type 2 that consists of at least \( N + 1 \) time steps and starts at some time \( t \). Then, by the pigeonhole principle, there must be two points in time \( t < t' < t'' \leq t + N + 1 \) such that \( q^t_{i,j} = q^{t''}_{i,j} \). Moreover, it must also hold that \( c^t_{j} = c^{t''}_{j} \), since otherwise \( a_i \) would move according to some non-zero travel vector (from time \( t' \) onwards) which would imply that \( S_j^i \) is not of type 2. This implies that if \( a_i \)’s subschedule would also continue at and after time \( t + N + 1 \) on an empty grid, then \( a_i \) would cycle through the same movement on and on, starting from time \( t' \). Hence, if there is a cell \( c \) that is visited by \( a_i \) in some state \( q \) in the (continued) movement after time \( t'' \), then there must also be a point in time before \( t'' \) (during \( S_j^i \)) at which \( a_i \) visits \( c \) in state \( q \). It follows from the definition of our schedule that \( S_j^i \) ends before time \( t'' \), yielding a contradiction to our assumption. \( \square \)

**Lemma 3.** Any subschedule \( S_j^i \) of type 1, where agent \( a_i \) ends in the same cell from which it started, consists of at most \( N(2N + 1) \) time steps. More generally, any subschedule \( S_j^i \) of type 1, where \( a_i \) ends in a cell of distance at most \( D \) from the cell from which it started, consists of at most \( N(2N + 1 + D) \) time steps.

**Proof.** We start by proving the special case where \( a_i \) ends in the same cell from which it started. Suppose for a contradiction that there is a subschedule \( S_j^i \) as described in the lemma that consists of more than \( N(2N + 1) \) time steps. Let \( t \) and \( u \) denote the points in time when \( S_j^i \) starts and ends, respectively. Since \( a_i \) does not encounter any other agent between time \( t \) and time \( u \), it behaves like a single agent on an empty grid between \( t \) and \( u \). In particular, there is a travel vector \((x, y)\) of agent \( a_i \) from time \( t + 1 \) to time \( u - 1 \) since \( N(2N + 1) - 1 > N \). For reasons of symmetry, we can assume w.l.o.g. that \( x > 0 \) and \( y \geq 0 \). Note that \( x = 0 = y \) is not possible since in that case \( a_i \) would cycle through the same (cyclic) movement over and over without meeting any other agent, which would imply that \( S_j^i \) is not of type 1. Let \( p \) be the travel period which, according to its definition, is at most \( N \).

Let \( q \) be the state whose second occurrence during \( S_j^i \) (excluding the state at the beginning of \( S_j^i \)) comes earliest. Let \( t' \) be the time when \( q \) occurs for the first time. Since \( t' \leq t + N \), we know that \( x_{t'}^i \geq x^i_j - N \). Now, as in each travel period \( a_i \) increases the \( x \)-coordinate of the cell it occupies by at least 1, it follows that at time \( t' + 2N \cdot p \) the \( x \)-coordinate of the cell \( a_i \) occupies is at least \( x^i_j + N \). Furthermore, since in each further travel period \( a_i \) would advance by at least one cell in (positive) \( x \)-direction in total and \( p \leq N \), after time \( t' + 2N \cdot p \) agent \( a_i \) will never have an \( x \)-coordinate of less than \( x^i_j + 1 \), i.e., it will never reach \( c^j \). But \( a_i \) also cannot have visited \( c^j = c^p_j \) between time \( t + 1 \) and \( t' + 2N \cdot p \) since \( t' + 2N \cdot p \leq t + N(2N + 1) \) and \( S_j^i \) consists of more than \( N(2N + 1) \) time steps. Thus, we obtain a contradiction, which proves the first lemma statement.
Figure 2: In Figure 2a, the travel vector, denoted by the black arrow, of the agent denoted by the red circle is determined by the movements during one cycle of state changes. Even though most movements performed in this cycle may not lead the agent along the travel vector, there has to be at least a translation of one movement from the original position within every $N$ time steps. Thus, after at most $D$ cycles, a destination in distance $D$ is reached. Furthermore, once the minimum distance between two agents gets large, as depicted in Figure 2b, only travel vectors parallel to a line with a certain slope lead from one far away agent to the other.

For the more general second statement, by an analogous proof we obtain that after time $t' + 2N \cdot p + D \cdot p$ agent $a_i$ will never have an $x$-coordinate of less than $x_i + 1 + D$, i.e., it will never reach $c_i$ then. But $a_i$ also cannot have visited $c_i$ between time $t + 1$ and $t' + 2N \cdot p + D \cdot p$ since $t' + 2N \cdot p + D \cdot p \leq t + N(2N + 1 + D)$ under the assumption that $S_i$ consists of more than $N(2N + 1 + D)$ time steps. Hence, this assumption must be false, and the lemma statement follows.

5.3 Traveling and Meeting

Having defined and studied the schedule, we now proceed with our lower bound proof as described in the high-level overview. The next lemma shows that for each distance there is a point in time after which the farthest two ants are never closer than this distance.

**Lemma 4.** For each distance $D$ there is a time $T$ such that at any time $t \geq T$ the largest pairwise distance of the three agents is at least $D$.

**Proof.** Suppose that the lemma statement is not true. Then there is an infinite sequence $T$ of points in time such that at each of these points in time the largest pairwise distance of the three agents is less than $D$. Since the distances of the agents are less than $D$ at all points in time from $T$ and the number of states the three agents can be in is finite, it follows that there must be points in time $t, t' \in T$ such that 1) each agent is in the same state at $t$ and $t'$, 2) $x_i - x_i = x_i - x_i$ and $y_i - y_i = y_i - y_i$ for all $i, j \in \{1, 2, 3\}, i \neq j$, and 3) the same agent is scheduled to move next. Since the agents are oblivious of the absolute coordinates of the grid, this implies that from time $t'$ on, the agents will repeat the exact behavior they showed starting at time $t$. (Note that we use here that the schedule following a configuration is uniquely determined by the above information.) Hence, at time $t' + (t' - t)$ the agents will again be in the exact same configuration and so on.
Define \((x, y) = (x^i_t - x^i_0, y^i_t - y^i_0, z^i_t - z^i_0, w^i_t - w^i_0)\), where \(i = 1\) (which implies that this equation also holds for \(i = 2, 3\)). Vector \((x, y)\) describes the total movement of each of the agents during each of the (repeating) time periods of length \(t' - t\). It follows that each cell that has not been explored by time \(t\) must be at distance at most \(t' - t\) from some cell that is obtained by adding a multiple of the vector \((x, y)\) to one cell from \(\{c^1_t, c^2_t, c^3_t\}\); otherwise it will never be explored. Since each such cell at distance at most \(t' - t\) (which is constant) must lie in a band of constant width and “direction” \((x, y)\) that contains \(c^1_t, c^2_t\) or \(c^3_t\), there are infinitely many cells that must have been explored before time \(t\). This yields a contradiction.

For any distance \(D\), we denote by \(T_D\) the smallest time \(T\) for which it holds that at any time \(t \geq T\) the largest pairwise distance of the three agents is at least \(D\). Please refer to Figure 2 for illustrations of properties of the travel vector and the increasing distances.

In the following we collect a number of useful definitions regarding the meetings of different agents. In particular, we distinguish between three different types of agents at times when one agent is traveling from another agent to the far-away agent whose existence is certified by Lemma 4.

**Definition 1.** For any \(t \geq 0\), we define the meeting set \(M_t\) as the set of agents that are not alone in the cell they occupy, at time \(t\). We call the infinite sequence \((M_0, M_1, \ldots)\) the meeting sequence. If for a subsequence \((M_t, M_{t+1}, \ldots, M_{t+i})\) of the meeting sequence it holds that \(i > 0\), \(M_t \neq \emptyset \neq M_{t+i} \text{ and } M_{t+i} = \emptyset\) for all \(0 < j < i\), then we call the pair \((t, t+i)\) a meeting pair. Now, let \((t, u)\) be a meeting pair such that \(|M_t| = 2 = |M_u|\) and \(M_t \neq M_u\). Then we call \((t, u)\) a travel meeting pair. Moreover, we call the (uniquely defined) agent \(a\) contained in \(M_t \cap M_u\) a traveling agent (for \((t, u)\)), the agent contained in \(M_t \setminus \{a\}\) a source agent and the agent contained in \(M_u \setminus \{a\}\) a destination agent.

In order to continue according to our high-level proof idea, we need a few helping lemmas that highlight properties of the previous definitions. We start with a lemma that shows an important property of the meeting sequence:

**Lemma 5.** Each of the three agents is contained in infinitely many of the \(M_t\) from the meeting sequence.

**Proof.** Suppose that there is an agent \(a_i\) that is not contained in infinitely many of the \(M_t\), i.e., there is a point in time \(u\) such that \(a_i \notin M_t\) for all \(t \geq u\). Then, starting from time \(u\), the exploration by the two agents \(a_r, r \neq i\) is entirely independent of the exploration by agent \(a_i\), since they never meet again. Hence, we obtain a contradiction analogously to the argumentation in the proof of Lemma 1.

Next, we study travel meeting pairs more closely. In Lemma 6, we present bounds on the number of subschedules of the different types of agents in the time frame given by a travel meeting pair, and examine the types of the subschedules. Afterwards, in Lemma 7, we bound the number of time steps between two subsequent travel pairs from above. In both cases, the results only hold from a large enough point in time onwards, but this is sufficient for our purposes since before that point in time only a constant number of cells were explored. Note that, in general, we do not attempt to minimize the dependence on \(N\) in our
bounds as showing the finiteness of certain parameters is, again, sufficient for our purposes. Instead we prefer to choose the simplest arguments that lead to the desired finiteness results, even if they augment the actual bound by a few factors of $N$.

**Lemma 6.** There is a point in time $T$ such that, for each travel meeting pair $(t, u)$ with $t \geq T$, the following properties hold:

1. The traveling agent for $(t, u)$ is scheduled exactly once (for a number of time steps) between time $t$ and time $u$.
2. The subschedule of the traveling agent is of type 1 and ends exactly at time $u$.
3. The source and the destination agent for $(t, u)$ are scheduled at most once (for a number of time steps).
4. If the source or the destination agent is scheduled, then its subschedule is of type 2.

**Proof.** Recall the definition of $T_D$ for any distance $D$. Let $T \geq T_{2N+1}$, and consider an arbitrary travel meeting pair $(t, u)$ with $t \geq T$ and traveling agent $a_i$. Observe that if the source agent is scheduled between time $t$ and time $u$, then its subschedules must be of type 2, because the source agent is not contained in the meeting set $M_u$. Hence, if $a_i$ is not scheduled at all between time $t$ and time $u$, then the source agent must be scheduled at most once (because of the specification of our schedule) which implies that its distance from $c_i$ at time $u$ is at most $N$, by Lemma 2. But since in this case $a_i$ and the destination agent meet at $c_i$ at time $u$, we obtain a contradiction to the fact that $T \geq T_{2N+1}$.

Thus, we know that $a_i$ is scheduled at least once between time $t$ and time $u$.

Now, assume for a contradiction that the first subschedule of $a_i$ between time $t$ and time $u$ is of type 2. This implies that if one would schedule $a_i$ on and on, it would repeat a state in the same (empty) cell after at most $N + 1$ time steps and then cycle through (a part of) the same movement it performed before. Hence, even if there are more subschedules for $a_i$ than one (between time $t$ and time $u$), it will never reach a cell that has a distance of more than $N$ from $c_i$. Since analogous statements hold for the source agent, we know that at time $u$ the distance between the source agent and the cell where $a_i$ and the destination agent meet is at most $2N$ which again contradicts our specification of $T$.

Thus, we know that the first subschedule of $a_i$ is of type 1. It follows that $a_i$’s subschedule ends exactly at time $u$ since the subschedule must end with $a_i$ meeting the destination agent, which also implies that $a_i$ is scheduled exactly once between time $t$ and time $u$. Moreover, the subschedules of the source and the destination agent (if they are scheduled at all between time $t$ and time $u$) must be of type 2 since $(t, u)$ is a (travel) meeting pair. Furthermore, by the nature of our schedule, the source and the destination agent must be scheduled at most once between time $t$ and time $u$.

**Lemma 7.** There is a point in time $T$ such that the following holds: If $(t, u)$ and $(t', u')$ are travel meeting pairs such that $T \leq t < t'$ and there exists no travel meeting pair $(t'', u'')$ with $t < t'' < t'$, then $t' - u \leq 2(N + 1)^7$. 

12
Proof. Observe that from the definition of a travel meeting pair it follows that $t' > u$. Set $T := T_{(N+1)^4}$ and let $t, u, t', u'$ be as described in the lemma. Without loss of generality, let $a_1$ and $a_2$ be the agents contained in $M_u$. Let $t_1 < t_2 < \cdots < t_k$ be exactly the points in time $t_j$ between $u$ and $t'$ for which $M_{t_j} \neq \emptyset$ holds. It follows that all $M_t$ are identical to $M_{u'}$.

We claim that $k < (2N^2 + 1)(N + 1)$. Suppose for a contradiction that $k \geq (2N^2 + 1)(N + 1)$. Then, there must be at least $2N^2 + 1$ indices $g \in \{1, \ldots, k\}$ such that $a_1$ or $a_2$ are scheduled to move at time $t_g$ since each subschedule of $a_3$ between time $u$ and $t'$ is of type 2 and hence consists of at most $N$ time steps, by Lemma 8. It follows that there must be some $1 \leq g < h \leq k$ such that $a_1$ and $a_2$ are in the exact same (pair of) states at time $t_g$ and at time $t_h$, and the same agent is scheduled next. This implies that $a_1$ and $a_2$ go through the same movement, that they executed between time $t_g$ and $t_h$, over and over again, starting from time $t_h$, until at least one of them encounters agent $a_3$. Recall that our schedule is oblivious, i.e., only depends on the current states and locations of the agents and the information which agent is scheduled next.

During this movement, i.e. anytime after $t_g$ (and also before), our agents $a_1$ and $a_2$ cannot move too far from each other as we show in the following: Due to the nature of our schedule, if an agent executes a subschedule of type 2 and then its next subschedules are again of type 2, then after at most $N + 1$ such subschedules the agent ended its subschedule at least twice in the same cell in the same state, by Lemma 2 and the specification of type 2 subschedules. Hence, between any two subschedules (of the agents $a_1$ and $a_2$) of type 1 after time $t_g$, there are at most $(N + 1)^2$ subschedules of type 2 (of those agents), as otherwise the two agents would repeat their combined states and locations and thus, never execute another subschedule of type 1 which is a contradiction to their cyclic movement. Now by Lemma 2 and Lemma 3 and the fact that after each subschedule of type 1 the agents $a_1$ and $a_2$ are in the same cell, it follows that the maximum distance of the two agents after time $t_g$ is at most $N(2N + 1 + D)$ where $D = N(N + 1)^2$. Hence, when one of the two agents encounters agent $a_3$, then the other is at distance at most $N(2N + 1 + N(N + 1)^2) < (N + 1)^4$. This contradicts the fact that $t \geq T_{(N+1)^4}$ and proves the claim.

From the above, we obtain the following picture: There are at most $k < (2N^2 + 1)(N + 1)$ subschedules of type 1 between time $u$ and $t'$ (since, when a subschedule of type 1 ends, the corresponding element from the meeting sequence is non-empty). Between any two subschedules of type 1 (and possibly before the first/after the last) there are at most $(N + 1)^2$ subschedules of type 2 of agents $a_1$ and $a_2$ and hence also not more than $1/2 \cdot (N + 1)^2 + 1$ subschedules of type 2 of agent $a_3$. Now, by Lemma 2 and Lemma 3 we obtain that $t' - u \leq ((2N^2 + 1)(N + 1) + 1)(3/2 \cdot (N + 1)^2 + 1)N + (2N^2 + 1)(N + 1)N(2N + 1 + D)$ where $D = N(N + 1)^2$. Hence, $t' - u \leq 2(N + 1)^7$.

Using Lemma 7 we show in the following that the information contained in two agents that meet already determines the times for the next travel meeting pair, independently of location and state of the third agent. This concludes our collection of helping lemmas.

Lemma 8. There is a point in time $T$ such that the following holds: For each $M_t$ in the meeting sequence with $t \geq T$, the information which agents are contained in $M_t$ and what states they are in at time $t$ uniquely determines whether there is
a $u > t$ such that $(t, u)$ is a travel meeting pair, and, if this is the case, which agent is the traveling agent for $(t, u)$.

**Proof.** Let $T$ be sufficiently large so that $T \geq 2(N+1)^7 + 1$ holds and Lemma 7 applies. Let $(t', u')$ be a travel meeting pair with $t' \geq T$, and let $t \geq u'$. By Lemma 5 such $t', u', t$ must exist. We start by observing that if $|M_t| \neq 2$, then there is no $u$ as described in the lemma. Thus, assume that $|M_t| = 2$, and denote the agents in $M_t$ by $a_i$ and $a_j$, where $i, j \in \{1, 2, 3\}, i \neq j$.

By our specification of $T$, we know that the distance at time $t$ between the third agent and the cell that $a_i$ and $a_j$ occupy is at least $2(N + 1)^7 + 1$. Hence, from time $t$ to time $t + 2(N + 1)^7$ neither $a_i$ nor $a_j$ encounters the third agent, which implies that the information which agents are contained in $M_t$ and what states they are in at time $t$ uniquely determines the behavior of agents $a_i$ and $a_j$ between time $t$ and time $t + 2(N + 1)^7$. Now, if there is a $t''$ with $t < t'' \leq t + 2(N + 1)^7$ such that $M_{t''} \neq \emptyset$ (i.e., $M_{t''} = M_t$), then it follows that there is no $u$ as described in the lemma. On the other hand, if there is no such $t''$, then all meeting sets between time $t + 1$ and $t + 2(N + 1)^7$ are empty, which implies that a $u$ as described in the lemma exists, by Lemma 6. Since the existence of a $t''$ as described above only depends on the behavior of $a_i$ and $a_j$ between time $t$ and time $t + 2(N + 1)^7$, the lemma statement follows. $\square$

### 5.4 The Travel Vector and a Modulo Operation

After collecting the above helping lemmas, we are now all set to formally prove the statements from our proof sketch. Before going through the statements one by one, let us for convenience define the notion of a travel: Let $(t, u)$ be a travel meeting pair. By Lemma 6 we know that the traveling agent for $(t, u)$ is scheduled exactly once between $t$ and $u$. We call the corresponding subschedule (or the movement during that subschedule) a travel. Recall the definition of travel vector and travel period. Note that a travel only has a travel vector (and period) if the traveling agent repeats a state (in empty cells) during the travel. Furthermore, observe that if a travel has a travel vector, then at least one entry of the travel vector is non-zero, due to the choice of our schedule.

We now prove the first of the mentioned statements, namely, that any travel vector after a certain point in time has the same slope.

**Lemma 9.** There is a point in time $T$ and a (possibly negative) ratio $r$ such that each travel starting at time $T$ or later has travel vector $(x, y)$ with $y/x = r$. For the sake of simplicity, assume that $r$ is set to $\infty$ if $x = 0$.

**Proof.** Let $T$ be sufficiently large so that Lemma 6 applies. Then, by Lemma 4 we know that any travel starting at time $T_N$ or later actually has a travel vector (and period).

Now, consider two travel meeting pairs $(t, u)$ and $(t', u')$ with $t < t'$ such that there is no travel meeting pair $(t'', u'')$ with $t < t'' < t'$. Let $t \geq T$ where $T$ will be specified later. For now, we just assume that $T$ is sufficiently large so that Lemma 6 and Lemma 7 apply. Let $(x, y)$, $(x', y')$ be the travel vectors for the travels corresponding to $(t, u)$ and $(t', u')$, respectively. Assume that $y'/x' \neq y/x$, where, again, we set the ratio to $\infty$ if the denominator is 0. Note that not both of $x$ and $y$ (or $x'$ and $y'$) can be 0. Let $c_0$ and $c_1$ be the cells at
Analogously, we obtain
\[ \text{Dist}(x, y) \]
which the travel with travel vector \((x, y)\) starts and ends, respectively, and \(c'_0\) and \(c'_1\) analogously for the travel with travel vector \((x', y')\).

By the characterization of the travel of a single agent and the fact that the travel period is always at most \(N\), we know that there are positive integers \(b\) and \(b'\) such that \(\text{Dist}(c_1, c_0 + b \cdot (x, y) \leq N\) and \(\text{Dist}(c'_1, c'_0 + b' \cdot (x', y') \leq N\). Moreover, by Lemma 2 and Lemma 6, the source agent for \((t, u)\) travels at most a distance of \(N\) between time \(t\) and \(u\) since its subschedule is of type 2 if the agent is scheduled at all. The same holds for the destination agent for \((t', u')\) between time \(t'\) and \(u'\). By Lemma 7, it follows that \(\text{Dist}(c_0, c'_1) \leq 2(N + 1)^7 + 2N\) (since the source agent for the first of the two travels is the destination agent for the second) and \(\text{Dist}(c_1, c'_0) \leq 2(N + 1)^7\). Combining our above distance observations, we also obtain \(\text{Dist}(c'_1, c_0 + b \cdot (x, y) + b' \cdot (x', y')) \leq N + 2(N + 1)^7 + N\), which together with \(\text{Dist}(c_0, c'_1) \leq 2(N + 1)^7 + 2N\) implies \(\text{Dist}(c_0, c_0 + b \cdot (x, y) + b' \cdot (x', y')) \leq 4(N + 1)^7 + 4N\).

Let \(D \geq N\) be some positive integer. We now require, additionally to the above requirements regarding \(T\), that \(T \geq T_D\). At the time when the first of the two considered travels starts there are two agents at \(c_0\) and \(c_1\) while the last agent is in distance at most \(N\) from \(c_0\). Hence, the distance between \(c_0\) and \(c_1\) is at least \(D - N\). This implies that \(b \cdot (|x| + |y|) \geq \text{Dist}(c_1, c_0) - N \geq D - 2N\).

Analogously, we obtain \(b' \cdot (|x'| + |y'|) \geq D - 2N\). Since \(x, y, x', y'\) are fixed, we can therefore make \(b\) and \(b'\) arbitrarily large by increasing \(D\). By increasing \(b\) and \(b'\), we can in turn make \(\text{Dist}(c_0, c_0 + b \cdot (x, y) + b' \cdot (x', y'))\) arbitrarily large, since \(y'/x' \neq y/x\) (which implies that there is an angle between the two vectors \((x, y)\) and \((x', y')\) that is not 0° or 180°). Hence, if \(D\) is sufficiently large, then the above inequality \(\text{Dist}(c_0, c_0 + b \cdot (x, y) + b' \cdot (x', y')) \leq 4(N + 1)^7 + 4N\) is not satisfied anymore, which shows that \(y'/x' = y/x\). This completes the proof.

Note that the magnitude \(D\) has to reach for this (in our proof by contradiction) depends on \(x, y, x', y'\). However, since the number of possible travel vectors of a single agent is bounded by the number of states in its finite automaton, we can simply derive a sufficiently large \(D\) for each of the finitely many possible combinations for \(x, y, x', y'\) and then choose a \(T\) that is larger than all of the respective \(T_D\).

Note that the exact value of \(r\) depends only on the finite automaton governing the behavior of the three agents. From now on, we denote the ratio whose existence is certified by Lemma 7 by \(r\). W.l.o.g., we can (and will) assume that \(r \geq 0\) (and that \(r \neq \infty\)), for reasons of symmetry. Recall that any travel vector has at least one non-zero entry.

The next step on our agenda is essentially to show that the state of an agent at the end of a travel does not depend on (the full information about) the vector between start and endpoint of that travel (and other parameters), but only on a reduced amount of information regarding this vector (and the other parameters). More specifically, the required information about this vector is the result of applying a certain modulo operation to the vector. We then proceed by showing that the information about 1) the states of the agents, 2) their relative locations after applying the modulo operation, 3) which agents shared a cell most recently, and 4) which agent is scheduled next, at the start of a travel, is enough to determine the exact same information at the end of the travel. Now, we benefit from the previous reduction of information due to our modulo operation in the sense that we can show that there are only constantly many combinations of
relative locations of the three agents (that can actually occur) after applying the modulo operation. This, in turn, implies that there are only constantly many possibilities for the whole aforementioned information tuple at the start and end of a travel, which will enable us to prove our main theorem.

We start by defining our modulo operation in Definition 2. Then we show a technical helping lemma, Lemma 10, which finally enables us to prove the aforementioned relation between the information tuple at the start and end of a travel in Lemma 11.

**Definition 2.** Let \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\} \) be the set of travel vectors that the agents can have if you let one of them explore the grid starting in an arbitrary state (which clearly is a superset of the actually occurring travel vectors in our multi-agent case). Let \( R \) be the subset of the above set that contains exactly the vectors \((x_j, y_j)\) that satisfy \(y_j/x_j = r\). From now on, denote by \( x \) be the least common denominator of the \( |x_j| \) from the vectors in \( R \) and set \( y := rx \). It follows that \((x, y)\) is a (possibly negative) integer multiple of any of the vectors from \( R \).

Now, let \( w, z \) be integers and let \( b \) be the smallest integer such that \( w+bx \geq 0 \). We define \((w, z) \pmod{(x, y)} := (w+bx, z+by)\). For two cells \((w', z'), (w'', z'')\), we define \((w'', z'') \cap (w', z') := (w'' - w', z'' - z') \pmod{(x, y)}\).

Note that Definition 2 ensures that for any \((w, z), (w', z')\) where \((w-w', z-z')\) is a multiple of \((x, y)\), we have that \((w, z) \pmod{(x, y)} = (w', z') \pmod{(x, y)}\).

**Lemma 10.** Let \( a \) be an agent, \( q \) a state from \( a \)'s finite automaton and \( c, c', c'' \) cells of the grid such that the following properties are satisfied:

1. \( \text{Dist}(c, c') \geq N \) and \( \text{Dist}(c'', c') \geq N \)
2. There is an integer \( b \) such that \( c'' - c = b \cdot (w, z) \), where \((w, z)\) is agent \( a \)'s travel vector if it starts in state \( q \).
3. If agent \( a \) starts in cell \( c \) in state \( q \) on an otherwise empty grid, then it arrives at \( c' \) after finite time.
4. If agent \( a \) starts in cell \( c'' \) in state \( q \) on an otherwise empty grid, then it arrives at \( c' \) after finite time.

Let \( q' \) denotes the state in which \( a \) arrives at \( c' \) (for the first time) when starting from \( c \) (in state \( q \)), and \( q'' \) the state in which \( a \) arrives at \( c' \) (for the first time) when starting from \( c'' \) (in state \( q \)). Then it holds that \( q' = q'' \).

**Proof.** If \( c = c'' \), then the lemma holds trivially, thus assume that \( c \neq c'' \). W.l.o.g., we can assume that \( b > 0 \), which implies that, if agent \( a \) starts in cell \( c \) in state \( q \) (say, at time \( t \)), then \( a \) arrives at some point in time \( u > t \) in cell \( c'' \) in state \( q \) (possibly \( a \) visited \( c'' \) before in some other state). Hence, if \( a \) does not visit cell \( c' \) between time \( t \) and time \( u \), then the lemma also holds since after arriving at \( c'' \) in state \( q \), \( a \) will perform the exact same movement as if it started in \( c'' \) in state \( q \).

Thus, consider the last remaining case, i.e., assume that \( a \) visits \( c' \) for the first time at some time \( t < t' < u \). W.l.o.g., we can assume that \( w \) and \( z \) are non-negative and \( w, z \geq 0 \). (Also recall that at least one of \( w \) and \( z \) is non-zero.) Let \( c_0, c_1, \ldots \) be the cells that \( a \) visits in state \( q \) at and after time \( t \), where \( c_0 \)

16
and \(c_k\), for some \(k > 0\), are the cells that \(a\) visits at time \(t\) and \(u\), respectively, i.e., \(c_0 = c\) and \(c_k = c''\). Observe that \(c_{j+1} = c_j + (w, z)\) holds for each \(j\). Denote the \(x\)-coordinates of \(c'\) and \(c_k = c''\) by \(x'\) and \(x''\), respectively. Since \(w \geq z\), it follows that \(\text{Dist}(c_j, c') \geq \text{Dist}(c'', c') \geq N\) for all \(j \geq k\) if \(x' \leq x''\), and \(\text{Dist}(c_j, c') \geq \text{Dist}(c'', c') \geq N\) for all \(0 \leq j \leq k\) if \(x' \geq x''\). Let \(h\) be the largest index such that \(a\) visits \(c_h\) in state \(q\) at or before time \(t'\). Then \(h < k\), and \(\text{Dist}(c_h, c') \leq N - 1\) since traveling from \(c_h\) (in state \(q\)) to \(c_{h+1}\) (in state \(q\)) takes \(a\) at most one travel period, so at most \(N\) time steps. If \(x'' \geq x''\), then we obtain a contradiction to our above observation, thus it follows that \(x' < x''\). But this implies \(\text{Dist}(c_j, c') \geq N\) for all \(j \geq k\) which in turn implies for all \(j \geq k\) that \(c'\) cannot be visited by \(a\) between visiting \(c_j\) (in state \(q\)) and \(c_{j+1}\) (in state \(q\)). Hence, \(a\) does not visit \(c'\) at or after time \(u\). Since \(a\) performs the exact same movement from time \(u\) onwards as if it would have initially started in \(c''\), it follows that \(a\) starting in \(c''\) in state \(q\) never visits \(c'\), which is a contradiction to our assumptions. Thus, this last remaining case cannot occur, which completes the proof.

\[\square\]

**Lemma 11.** Let \((t, u)\) be a travel meeting pair. Consider the tuple \(Q_t := (q^1_t, q^2_t, q^3_t, c^1_t \oplus c^2_t, c^3_t \oplus c^2_t, a^{\text{next}}_t, M_t)\), where \(a^{\text{next}}_t\) denotes the agent that is scheduled at time \(t\). There is a point in time \(T\) such that the following holds: If \(t \geq T\), then \(Q_t\) uniquely determines the tuple \(Q_u = (q^1_u, q^2_u, q^3_u, c^1_u \oplus c^2_u, c^3_u \oplus c^3_u, c^1_u \oplus c^2_u \oplus c^3_u, a^{\text{next}}_u, M_u)\).

**Proof.** Let \(T\) be sufficiently large so that \(T \geq T_{2N+1}\) holds and Lemma 6, Lemma 8, and Lemma 9 apply. Let \((t, u)\) be a travel meeting pair with \(t \geq T\). By Lemma 8, we know that \(Q_t\) uniquely determines which agent is the traveling agent for \((t, u)\), which in turn uniquely determines \(M_u\). Furthermore, by Lemma 6, we know that the last agent that is scheduled before time \(u\) is exactly the traveling agent, which uniquely determines \(c^{\text{next}}_u\).

Moreover, the information which agent is the traveling agent together with the information which agent is scheduled at time \(t\) (i.e., \(a^{\text{next}}_t\)) uniquely determines which agents are scheduled between time \(t\) and time \(u\) (and all of them are scheduled only once, possible for multiple subsequent time steps, if they are scheduled at all). Since no two agents meet between time \(t\) and time \(u\), it follows that the states of the source agent and the destination agent at time \(u\) are uniquely determined by their states at time \(t\) (and the information which two agents are contained in \(M_t\)). Here we use that the subschedules of those agents (if they are scheduled at all) are of type 2, according to Lemma 6.

Similarly, the exact vectors (possibly the vector \((0, 0)\)) by which the source agent and the destination agent move are uniquely determined by their states at time \(t\). By Lemma 8, the subschedule of the traveling agent is of type 1 and ends in the cell that is occupied by the agent not contained in \(M_t\). Since addition and our modulo operation behave nicely (or, more specifically, because \(((w, z) + (w', z')) \pmod{(x, y)} = ((w, z) \pmod{(x, y)} + (w', z')) \pmod{(x, y)}\) for all integers \(w, z, w', z'\)), it follows that \(c^1_u \oplus c^2_u, c^1_u \oplus c^3_u, c^2_u \oplus c^3_u\) are uniquely determined by the vectors by which source and destination agent move (combined with the information contained in \(Q_t\)), and thereby also by \(Q_t\).

It remains to show that the state of the traveling agent at time \(u\) is uniquely determined by \(Q_t\). Denote the traveling agent by \(a_t\). Since \(t \geq T_{2N+1}\), the
We now conclude our lower bound proof with Theorem 1. Roughly speaking, Theorem 1. Three agents are not sufficient to explore the grid. while all other requirements for Lemma 10 are still satisfied. This completes the proof.

Now, let \( c, c' \) be cells with a distance of at least \( N + 1 \) to the location \( c' \) of the destination agent at time \( t' \) and assume that \( c' \odot c = c' \odot c'' \). Then, according to the definition of our modulo operation, \( c'' - c \) is a (possibly negative) integer multiple of \( (x, y) \), and thus also of the travel vector of \( a_i \), by Lemma 9. Thus, by Lemma 10 it follows from the above that \( q'' \) is uniquely determined by \( Q_t \). Note that although \( a_i \) may not be alone in its cell at the time its subschedule starts, we can still apply Lemma 10 since after the first step of \( a_i \) it is alone in its cell while all other requirements for Lemma 10 are still satisfied. This completes the proof.

Note that the exact travel vector of \( a_i \) is also uniquely determined by \( Q_t \), by Lemma 8.

5.5 Three Agents Are Not Sufficient

We now conclude our lower bound proof with Theorem 1. Roughly speaking, Lemma 11 certifies that the behavior of the agents between any two subsequent occurrences of the same fixed information tuple \( Q_t \) is reasonably similar. Since there are only finitely many different \( Q_t \) that actually occur, it follows that the behavior of the agents loops, in a very informal sense. From this, we can derive a contradiction to the assumption that all cells are explored.

**Theorem 1.** Three agents are not sufficient to explore the grid.

**Proof.** Suppose for a contradiction that three agents suffice to explore the grid. From the definition of a travel meeting pair, it follows that there are points in time \( t_1 < u_1 \leq t_2 < u_2 \leq t_3 \leq \ldots \) such that \( (t_j, u_j) \) is a travel meeting pair for any \( j \geq 1 \) and for every travel meeting pair \( (t', u') \) there is a \( j \geq 1 \) with \( t' = t_j \) and \( u' = u_j \).

Recall the definition of \( Q_t \) in Lemma 11. Let \( T \) be sufficiently large so that \( T \geq T_1 \) holds (where \( T_1 \) is just \( T_D \) for \( D = 1 \)) and Lemmas 6, 7, 9 and 11 apply, and let \( k \) be an index such that \( t_k \geq T \) and there is a \( h > k \) with \( Q_{t_k} = Q_{t_h} \). Such a \( k \) must exist since there is only a finite number of tuples of the general form \( Q_t \) (after time \( T \)) and the number of traveling meeting pairs is infinite, by Lemma 3. Note that the finiteness of the number of tuples, in particular the finiteness of the (combinations of the) relative locations of the agents modulo \( (x, y) \), relies on the fact that the possible travel vectors after time \( T \) are restricted by Lemma 3.

Consider the sequence \( (t_h, u_h), (t_{k+1}, u_{k+1}), \ldots, (t_b, u_b) \) of traveling meeting pairs, where \( b \) is the smallest index such that \( h > k \) holds, \( h - k \) is even, and \( Q_{t_h} = Q_{t_b} \). We examine the cells that are explored by the source agent for \( (t_h, u_h) \) between time \( t_h \) and \( t_{k+1} \), by the destination agent for \( (t_{k+1}, u_{k+1}) \) (which is the same as the aforementioned source agent) between time \( t_{k+1} \) and \( t_{k+2} \), and by the traveling agent for \( (t_{k+1}, u_{k+1}) \) between time \( u_{k+1} \) and \( t_{k+2} \). Then we iterate this examination, in each iteration increasing the indices by 2, and stop at time \( t_h \). We say that the cells explored in the described way are explored during even
explorations. In the first iteration, the explored cells are uniquely determined by the tuple \( Q_{t_k} \) and the exact location of the source agent for \((t_k, u_k)\) at time \( t_k \). In the next iteration by the tuple \( Q_{t_k+2} \) and the exact location of the source agent for \((t_k+2, u_{k+2})\) at time \( t_{k+2} \), and so on. The tuples \( Q_{t_k+2}, Q_{t_k+4}, \ldots \) are all uniquely determined by \( Q_{t_k} \), and the locations of the respective source agents at times \( t_{k+2}, t_{k+4}, \ldots \) are all uniquely determined by \( Q_{t_k} \) and the location of the source agent for \((t_k, u_k)\) at time \( t_k \). Hence, the cells explored during an even exploration between time \( t_k \) and \( t_h \) are uniquely determined by \( Q_{t_k} \) and the location of the source agent for \((t_k, u_k)\) at time \( t_k \).

We obtain the following picture: The location of the source agent for \((t_k, u_k)\) at time \( t_k \) together with \( Q_{t_k} \) uniquely determines both \( Q_{t_k} \) and the location of the source agent for \((t_h, u_k)\) at time \( t_h \), which, in turn, uniquely determine \( Q_{t_h+h-k} \) and the location of the source agent for \((t_h+(h-k), u_{h+(h-k)})\) at time \( t_{h+(h-k)} \), and so on. Hence, there is a vector \((w, z)\) such that the locations of the respective source agents at times \( t_k, t_h, t_{h+(h-k)}, t_{h+2(h-k)}, \ldots \) are \( c, c+(w, z), c+2(w, z), \ldots \), where \( c \) denotes the cell occupied by the respective source agent at time \( t_k \). Moreover, since the number of cells explored during an even exploration between time \( t_k \) and \( t_h \) (and similarly between time \( t_h+j(h-k) \) and \( t_h+j+1(h-k) \) for each \( j \geq 0 \)) is uniquely determined by \( Q_{t_k} \), we obtain that there is a constant \( L \) such that each cell explored during an even exploration has a distance of at most \( L \) to some cell of the form \( c+j' \cdot (w, z) \) with \( j' \geq 0 \).

Moreover, by Lemmas 2, 3, 6, 7, and the definition of even explorations, we know that each explored cell is close to the travel of a traveling agent, i.e., there is a constant \( L' \) such that each cell explored at or after time \( t_k \) has a distance of at most \( L' \) to some cell of the form \( c+j'' \cdot (x, y) \), where \( j'' \geq 0 \) and \( c \) is a cell explored during an even exploration.

Combining our observations and adding the fact that only a constant number of cells are explored up to time \( t_k \), it follows that there is a constant \( L'' \) such that each cell explored by the agents has a distance of at most \( L'' \) to some cell of the form \( c+j' \cdot (w, z) + j'' \cdot (x, y) \) with \( j', j'' \geq 0 \). Hence, we can draw a line in the grid such that all explored cells are to one side of the line, yielding a contradiction to the assumption that three agents suffice to explore the grid. \( \blacksquare \)

6 Four Non-compass Agents Do Not Suffice

The main goal of this section is to show a lower bound that states that four agents that are not equipped with a compass cannot explore the grid. To use the lack of compass to our advantage, we introduce methods to mirror the port numbering in a way that if port \( i \) leads to port \( j \) before the mirroring, the same holds after the mirroring. Specifically, assume that all ports are globally consistent, i.e., ports 1, 2, 3, and 4 correspond to north, west, east, and south, respectively. Furthermore, ports 1, 2, 3, and 4 lead to ports 4, 3, 2, and 1, respectively.

Mirroring vertically above a horizontal line is defined as mirroring ports 2 and 3 of all cells above the line. Mirroring vertically below a horizontal line and horizontally on the left and right of a vertical line are defined analogously. Consider a diagonal with slope 1. When we mirror above the diagonal, we exchange ports 1 and 2 along the diagonal. Every cell below the diagonal is left unchanged and in the ports above the diagonal, we exchange ports 1 and 2 and ports 3 and 4. This way, every port leads to the port with the same number as
before. Mirroring below a diagonal is defined analogously. Refer to Figure 3 for an illustration.

**Lemma 12.** Assume a globally consistent port numbering. Then, after mirroring vertically above or below a horizontal line, horizontally left or right of a vertical line, or diagonally above or below a diagonal line, the ports 1, 2, 3, and 4 still lead to ports 4, 3, 2, and 1, respectively.

**Proof.** It is easy to verify that since all mirrorings are symmetric, any two adjacent cells where all ports are exchanged or remain the same, satisfy the lemma statement. Consider then two adjacent cells \( c \) and \( c' \) in the case of a vertical mirroring, where only \( c' \) is mirrored. Since the vertical mirroring is performed below or above a horizontal line, cells \( c \) and \( c' \) must be connected through ports 1 and 4. These ports are not affected and thus, the claim follows for this case. The horizontal case is analogous.

For the diagonal case, consider two adjacent cells \( c \) and \( c' \) that are both affected and let us assume that these cells are connected through ports 1 and 4. According to the definition of the mirroring, either both ports are changed (to 2 and 3, respectively) or they remain unchanged. The remaining case is shown analogously and thus, the claim follows.

### 6.1 Agents Cannot Travel Alone Without a Compass

In the following lemma, we show that an agent without a compass cannot travel alone. Assuming that this agent has to travel a long distance without any agents nearby, we can use mirroring to change the port numbering, and thereby changing the travel vector, which leads to not meeting any agent anymore. We will use the schedule defined in the three agent case, adjusted so that the fourth agent is always scheduled after the other three. Please refer to Figure 4 for an illustration of the following lemma.

**Lemma 13.** Assume that agent \( a \) is scheduled until \( a \) meets another agent and \( a \) has a non-zero travel vector. Additionally, assume that the distance from \( a \) to any other agent is more than \( 16N^3 + 12N \). Then, there is a mirroring operation that changes the port numbering outside of the view (i.e., \( N \) hops) of \( a \) in a way that agent \( a \) never meets any other agent again.
Figure 4: Let $a$ be an agent traveling according to a horizontal travel vector towards $a'$. To find a suitable spot for the diagonal below which we mirror, denoted by the dashed black line, we find intersections of vertical and diagonal lines that could potentially lead to some agent, denoted by the blue and red areas. Given that the band according to which $a$ travels is long enough, we can always guarantee the existence of a safe subband, denoted by the black area.

Proof. Similarly to the previous proofs we will be generous with our estimates for the sake of simplicity. Given a globally consistent orientation, $a$ has to be contained in a band $b$ of width $2N$ drawn on a line parallel to the travel vector going through the cell that $a$ currently occupies. We call a segment of band $b$ of length $\ell$ a subband of $b$ of length $\ell$.

Since the locations of the three other agents are fixed as long as $a$ is scheduled, we observe that the intersection of $b$ and any set of bands of width $2N$ perpendicular to $b$ containing some (fixed) agent $a' \neq a$, is contained in some subband of length $4N$. It follows that the intersections of all bands of width $N$ are contained in three subbands of length $4N$.

Furthermore, consider a band of width $2N^3$ that is parallel to the diagonal vector, i.e., $(1,1)$. Since the width of $b$ is less than $2N^3$, the intersection of such a band with $b$ is contained in a subband of length $4N^3$. Again, it follows that the intersection of $b$ with diagonal bands of width $2N^3$ that contain some agent $a' \neq a$ can be covered with three subbands of length $4N^3$. Finally, since the distance to any agent is more than $16N^3 + 12N$, we get that there has to be a subband $s^*$ of length $4N^3 \geq 2N^3 + 2N$ that does not intersect any of the aforementioned subbands.

We divide the rest of the proof into two cases. First, consider the case where $x = 0$ in the travel vector. We now mirror the port numbering diagonally through the middle of $s^*$. Once $a$ reaches the middle of $s^*$, two things can happen. First, assume that $a$ crosses the diagonal at least $N$ times, which implies that it has entered one diagonal cell twice in the same state. It is easy to verify that this can take at most $N^3$ time steps. It follows, that the travel vector is parallel to the diagonal (with respect to the new port numbering) and $a$ will never get further than $N$ steps from the diagonal. Since, no agent $a' \neq a$ is contained in a band of width at least $2N$ around the diagonal, $a$ will never meet any other agent.

Assume then that after at most $N^3$ steps, the agent never returns to the diagonal. The travel vector must be perpendicular to the old travel vector, implying that $a$ never meets any other agent, since no agent $a' \neq a$ is contained in a band perpendicular to $b$ of width $2N$ that intersects $s^*$.

In the case of a travel vector $(x,y)$, where both $x$ and $y$ are non-zero, we can use an argumentation analogous to above, but instead of a diagonal mirroring
use a horizontal (or vertical) mirroring in a long and broad enough subband of \( b \).
The claim follows.

6.2 Reduction from the Case of Three Ants

Given the fact that no agent can travel alone, we get roughly speaking the
property that at any given time there are at least two agents close to each
other. Otherwise, one agent would be forced to travel alone. Note that we can
apply Lemma 13 without assuming that the port numbering inside an agent’s
view is globally consistent: we simply choose the port numbering outside of
the agent’s view to be globally consistent and obtain that the agent can never go far
away from where it is (through the globally consistent area) without getting lost
according to Lemma 13.

This observation leads us to chase a simulation argument, where we simulate a
pair of nearby non-compass agents with one compass agent. With this simulation,
we can derive a three agent protocol that explores the grid under the schedule
from the previous section, from a correct four agent protocol. Since we have
shown that such a three agent protocol cannot exist, we obtain a contradiction.

According to our above observations, we obtain the following observation.
Note that since we can always do renaming of agents during meetings, we can
assume that the agents never change roles, i.e., the nearby agents remain the
same throughout the execution.

**Observation 2.** For constant \( \ell \), two fixed agents must always stay within
distance \( \ell \).

Consider the protocol \( \Pi_4 \) that finds the treasure against any schedule with
four agents without a compass. Let \( \mathcal{S} \) denote the schedule according to which
\( \Pi_4 \) visits the most state/relative location combinations within up to distance \( D \)
and let \( t \) be the point in time after which all of those combinations are assumed.
Then, no matter how we continue schedule \( \mathcal{S} \), it cannot be the case that any more
state/relative location combinations up to distance \( D \) are assumed. Otherwise,
we would obtain a combination twice, which implies that we can repeat the same
movement on and on, which yields a contradiction to the correctness of \( \Pi_4 \).

**Observation 3.** For each distance \( D \) there is a time \( T \) such that at any time \( t \geq T \) the largest pairwise distance of the four agents is at least \( D \).

Furthermore, we get that the agents cannot be split into groups of two that
are far apart. They cannot recover from this situation, since either one group of
two agents has to move to the other, breaking the minimum distance bound, or
one agent would have to travel alone.

Given that \( \Pi_4 \) is deterministic, described by a finite automaton, and holds
against any schedule, the three agents can initialize their protocol as follows.
First, one agent (hardcoded) explores every cell within (constant) distance \( T \) and
returns to the origin. Then, consider the four agent schedule \( \mathcal{S} \) from above and
determine the set of locations that the four agents occupy at time \( T \) according
to \( \mathcal{S} \). Note that by Observation 2, we get that at least one pair of agents have to
be close to each other. One of the three agents then chooses to simulate this pair and assumes the location of one of them, chosen arbitrarily. The other two agents assume the locations of the remaining two non-compass agents. At this point in time, the four agent schedule that is simulated, is implicitly determined by the three agents schedule and the three agent protocol explained in the following.

**Information Exchange** At all times, we have one of the three agents, say \(a\), simulating two of the four agents. Abstractly, this is implemented by encoding the states and the locations of the two agents into one state. Thereby, once some other agent meets \(a\), they can infer if and in what state they would have met either one of the simulated agents.

Let \(A_a = (q_a, x, y)\) be a tuple, where \(q_a\) is some state of agent \(a\) and \(x\) and \(y\) are coordinates relative to some cell. Note that in here, \(a\) stands for the agent to be simulated. Let \((A_a, A_{a'}, \{0, 1\})\) be a tuple where the states of two agents and their coordinates, relative to some cell, are encoded, in addition to a boolean value. Now, the agent \(\alpha\) who is simulating two agents, encodes such a tuple in all of its states, describing which agents he is simulating, in what states they are, and what their current locations are with respect to the location of \(\alpha\). In the case of the other agents, this tuple only contains the boolean value and the state and the location of the one agent they are simulating.

Since we have a free choice for the schedule for the four agents, we construct it implicitly to fit our needs for the simulation. Let \(N\) be an upper bound on the number of states in \(\Pi_4\). Consider now the schedule \(S\) for the three agents (introduced in Section 5) and let our protocol for the three agents be the following. Let \(\alpha\) be some agent that is simulating (one) agent \(a\). Every time agent \(\alpha\) is scheduled, it visits every cell in its \(2N\)-hop neighborhood to see whether there is any other agent nearby. During this time, \(\alpha\) turns the bit in the tuple \((A_a, \{0, 1\})\) to true, which stands for “currently checking”. In case any agent senses another agent in the checking state, then the agent remains in the cell ending its subschedule in one time step. This way, we allow the checking agent to finish the check it is currently performing. Note that by Observation 3 and due to our initialization procedure, we know that it can never be the case that all of the three agents are within a short distance during the simulation. Therefore, only one other agent can be found during this procedure.

After the neighborhood check, agent \(\alpha\) simulates one action of agent \(a\) according to \(\Pi\). During the simulation of a step, \(\alpha\) also enters the cell that the simulated agent \(a\) enters. Once the action is executed, \(\alpha\) repeats the check and performs another action, and so on. After a finite amount of such repetitions, the simulated agent \(a\) will either reach the same cell and state combination again, or meet another agent. Otherwise, \(a\) would have to be in a situation analogous to Case 3 of the three agents schedule, which is not possible. In the former case, \(\alpha\) will also reach a cell and state combination again and thus, the subschedule of another agent begins according to the three agent schedule we are using.

In the latter case, we have to be slightly more careful. If the cell, where \(a\) meets another simulated agent \(a'\), is not physically occupied by the agent \(\beta\) simulating \(a'\), then \(\alpha\) has to inform \(\beta\) about the fact that this meeting is taking place. Since \(\alpha\) collected the information about the whereabouts of \(\beta\) during the checking, \(\alpha\) can travel to the cell of \(\beta\) and turn off the checking flag. Thus, the subschedule of \(\alpha\) is finished and once \(\beta\) is scheduled, it can learn about the
meeting from the state of $\alpha$. For the case of simulating two agents $a_1$ and $a_2$, we assume that the schedule chosen for $\Pi_4$ always schedules $a_1$ and $a_2$ one after the other, before other agents. Thus, $\alpha$ can simulate the two agents in an analogous fashion, just taking turns between $a_1$ and $a_2$.

**Lemma 14.** If four agents without a compass suffice to explore the grid, then three agents with a compass suffice to explore the grid.

**Proof.** Let $\Pi_4$ be the protocol that allows the four non-compass agents to explore the grid. Now, we show how three compass agents can explore every cell explored by $\Pi_4$. Consider some time $\tau$ promised after which the largest pairwise distance is at least $D$ given by Observation 3 and assume that some pair of agents always stay within distance $\ell << D$, as promised by Observation 2.

Now, consider the simulation procedure described before and assume that the four agents being simulated start their execution from time $\tau$ on. Note that every cell explored by $\Pi_4$ until time $\tau$, is also explored by the three agents due to the specification of our initialization procedure. Given that the simulation method always visits all the cells visited by the simulated agents, the two things left to show are that, 1) every agent in the three agent protocol has all the information to simulate the corresponding agent(s) in the four agent protocol, and 2) for every agent in the four agent protocol, infinitely many steps are simulated. Otherwise, the four agent schedule that is simulated would not adhere to the specifications of a schedule.

For the first property, the information needed to simulate a step of the four agent protocol is the state and the location of the simulated agent, and the information of what other agents are in the same cell as the simulated agent, and in what states they are. Since the simulating agent always knows the location(s) of the agent(s) it simulates, and before simulating a step, it gathers the data about any nearby simulated agent(s) during the checking phase, we get the first property.

For property 2), note that in the three agent schedule, any agent gets scheduled infinitely often. Furthermore, once agent $\alpha$ starts the checking phase, it will execute a simulation step latest in the following subschedule of its own: If the subschedule of $\alpha$ is stopped by meeting another agent during the checking, the other agent will end its subschedule in at most one time step letting $\alpha$ finish the simulation.

Thus, the only obstacle could be that some agent $\alpha$ will be prevented from starting a checking phase by, always at the start of its subschedule, sharing its cell with an agent $\beta$ performing the checking. However, since the simulation performed by $\beta$ lasts until the simulated agent(s) either meet(s) some agent or enter(s) the same state cell combination again, it has to be the case that after a finite amount of time steps, either the distance between $\beta$ and $\alpha$ is larger than the checking radius or at least one of the agents simulated by $\beta$ meets one of the agents simulated by $\alpha$ upon which the subschedule of $\beta$ ends. This finishes the proof.

The following theorem follows by combining Lemma 14 and Theorem 1.

**Theorem 4.** Four asynchronous agents do not suffice to explore the grid in the port numbering model.
7 Five Agent Compass-Free Protocol

In this section, we show a matching upper bound to the lower bound result yielded by Theorem 4. The main underlying technique is to locally simulate a compass using two agents and thus, being able to move to some specific direction. Roughly speaking, the idea is to encode one of the cardinal directions in the relative positions of the agents and after every movement, ensure that this encoding is not broken. Note that given the lack of compass, the agents do not know which of the actual cardinal directions they have encoded. The difficulty lies in the lack of “proper” local vision, i.e., the two agents have to ensure that they can always find their way back to their companion using the very limited memory.

Throughout, our protocols can be implemented asynchronously. To break the ties of which agent performs their actions in which order, we can put encode flags to the agents’ states that indicate in which part of their current subroutine they are. Our protocols ensure that the agents that might influence each others routines are always nearby. Therefore, every agent can also check before starting its routine check if the other agents are finished with their routines. Furthermore, to check whether they have arrived to the immediate vicinity of another agents, they can (to be on the safe side), always visit their neighboring cells after every (sub)routine. For the sake of readability, these checks are omitted and assumed to be implicitly in the protocols.

We begin by introducing a simple routine that allows two agents $a$ and $a'$ positioned in cells $(x, y)$ and $(x, y - 1)$ to move to cells $(x, y + 1)$ and $(x, y)$. Note that given this configuration, agents $a$ and $a'$ have implicitly encoded to which direction is “south”. They would not be able to distinguish this configuration from the configuration where their locations are reversed (or horizontally aligned for that matter). The same routine can be implemented to any cardinal direction analogously. For the simplicity of presentation, we introduce some convenient subroutines listed below.

**Local Search** Subroutine LocalSearch($k$) stands for a local search that is a building block for the other operations. Subroutine LocalSearch($k$) is a simple DFS type search where the agents visit every cell (some perhaps multiple times) in a constant distance $k$ and finds its way back by storing the visited ports in its memory. It easy to verify that this can be implemented by a single agent in our model.

For the following operations, we assume that agents $a$ and $a'$ are within at most constant distance $k$ from each other and that $k$ is known to the agents.

**Swap** Operation Swap($a, a'$) swaps the locations of agents $a$ and $a'$. Agent $a$ finds the location of agent $a'$ by executing LocalSearch($k$) and stores a port sequence leading to $a'$ in its memory. Then, $a$ returns to its starting location and once the starting location is reached, assumes a special state that indicates that the original location is reached. Agent $a'$ performs LocalSearch($k$) (possibly many times) until it finds agent $a$. Then, agent $a$ can simply walk to the original location of $a'$ using the port sequence it stored.
Distance Assume agents $a$ and $a'$ are within distance $k$ bounded by a constant. The distance subroutine measures $\text{Dist}(a, a')$ which can be achieved by executing $\text{LocalSearch}(k)$.

Around For subroutine $\text{Around}(a', a)$, we assume that agents $a$ and $a'$ are within at most two hops from each other. Consider first the case that the distance is one and assume that $a$ occupies $(x, y)$ and $a'$ occupies $(x, y - 1)$. The goal is to move agent $a'$ to cell $(x, y + 1)$, i.e., “move around” agent $a$. Note that all other cases where the distance between the agents is 1 work analogously.

First, agent $a'$ performs $\text{LocalSearch}(1)$ to figure out which port leads to agent $a$, say port 1. Then, $a'$ performs $\text{LocalSearch}(3)$ to find a port sequence $S$ that leads to $a$ that does not involve port 1 from cell $(x, y - 1)$. Note that moving according to $S$ from $(x - 1, y)$ cannot involve entering cell $(x + 1, y)$, since the distance from $(x - 1, y)$ to $(x + 1, y)$ is greater than 2 through any other port than port 1. Now, $a'$ moves according to $S$ to a cell adjacent to $a$. Note that agent $a'$ does not know which of the two possible cells adjacent to $a$ it has entered.

Assume w.l.o.g., that the port leading to $a$ from the current location of $a'$ is port 1 and the port from which $a'$ entered its current location is port 2. The last step of $a'$ is to execute $\text{LocalSearch}(3)$ to find a port sequence that does not involve ports 1 and 2 (from the current cell of $a'$) and leads to the cell occupied by $a$, and move two steps according to this port sequence, to the cell adjacent to agent $a$. Agent $a'$ has now entered cell $(x + 1, y)$. Please refer to Figure 5 for an illustration.

The case where the initial distance is two is almost analogous. Assume that, initially, agent $a'$ occupies the cell $(x - 1, y - 1)$. The goal is now to move agent $a'$ diagonally over $a$ to the cell $(x + 1, y + 1)$. Agent $a'$ begins by finding a port that leads to a cell adjacent to $a$, say $(x - 1, y)$ by executing $\text{LocalSearch}(2)$. Similarly to the case above agent $a'$ can find its way to the cell $(x, y + 1)$, store the port from which it entered and figure out which port leads to the cell occupied by $a$. As the last step, agent $a'$ performs a $\text{LocalSearch}(2)$ from the ports not leading back nor to agent $a$ and can derive that cell $(x + 1, y + 1)$ is the one from which $\text{LocalSearch}(2)$ leads to $a$. All other diagonal cases work analogously.

Now, we are finally ready to introduce the movement routines that allow a pair of agents to move towards a desired cardinal or intermediate direction. Similarly to before, the agents are not aware of the actual cardinal direction, but we can fix one for the sake of simplicity. Assume that agents $a$ and $a'$ occupy cells $(x, y)$ and $(x - 1, y)$, respectively. The goal is to move agent $a$ to cell $(x + 1, y)$ and agent $a'$ to cell $(x, y)$.

Movement Given the subroutines above, it is now easy to implement a movement protocol for a pair of agents towards any cardinal or intermediate direction. More precisely, assume that agents $a$ and $a'$ are aligned according to some cardinal or intermediate direction, say southwest, i.e., agent $a$ is in some cell $(x, y)$ and agent $a'$ in the cell $(x + 1, y + 1)$. Now, agent $a'$ first performs $\text{Around}(a', a)$, that leads agent $a'$ to the cell $(x - 1, y - 1)$. Performing $\text{Swap}(a, a')$ leads agent $a$ to the cell $(x - 1, y - 1)$ and agent $a'$ to the cell $(x, y)$. In other words, agents $a$ and $a'$ have taken a step towards southwest. We say that agents $a$ and $a'$ move towards southwest when they take consecutive steps towards southwest.
Figure 5: To execute Around($a', a$), agent $a'$, denoted by the black circle, finds out which port leads to agent $a$, denoted by the red circle. Excluding this port, agent $a'$ performs LocalSearch(2) in all adjacent cells. The LocalSearch(2) performed one step to the north of the location $a'$, denoted by the red cells, cannot lead to $a$. Therefore, $a'$ finds a cell either to the east or to the west from $a$. The subroutine is finished by performing an analogous trick.

Figure 6: Illustration of a turn at north corner. First in Figure 6a agent $a'$ finds the cell east of $a$. Then in Figure 6b agents $a$ and $a'$ move once towards west. Finally, the routine is finished in Figure 6c by agent $a'$ and NorthGuide moving once towards north. The resulting positions are illustrated in Figure 6d.

All other directions work analogously.

7.1 Exploring the Grid

Given the routines introduced above, the next step is to design a protocol that allows the agents to explore every cell of the grid. Our protocol follows closely the ideas introduced in the work by Emek et. al. [16]. The basic idea is to mark the corners of a triangle in the grid by three agents, referred to as guides. Then, the two remaining agents, referred to as explorers move from corner to corner, additionally always expanding the triangle, while making sure that the next corner is reachable by moving along a straight line.

Even though the basic idea we apply is very similar to the previous work, we face the challenge of maintaining some sort of sense of direction while updating the triangle and moving between the corners. While the above routines allow us to perform movement between the corners, we have to be careful when expanding the triangle. Thus, the last tool we need is an update routine that ensures that the guides always wait for the explorers in some carefully chosen cell.
Figure 7: Illustration of routines executed to accomplish the turn at west corner.

Figure 8: Illustration of routines executed to accomplish the turn at east corner. The arrow accompanied with number 1 indicates that port 1 leads to the corresponding cell.

**Turn Southwest at North Corner**  Assume that agent $a$, $a'$ and NorthGuide reside in cells $(x, y + 1)$, $(x + 1, y)$, and $(x, y)$, respectively. The goal is to get agent $a$ to cell $(x - 1, y + 1)$, agent $a'$ to $(x, y + 2)$, and NorthGuide to $(x, y + 1)$. To achieve this, agent $a'$ begins by finding the cell in distance one adjacent to $a$ and not occupied by NorthGuide by executing LocalSearch(2). Then, the agents execute Around($a'$, $a$) and Swap($a$, $a'$). Finally, NorthGuide executes Around(NorthGuide, $a'$) and then Swap(NorthGuide, $a'$) finishes the routine. See Figure 6 for an illustration.

**Turn East at West Corner**  Assume that agents $a$, $a'$ and WestGuide reside in cells $(x - 1, y)$, $(x, y + 1)$, and $(x, y)$, respectively. The goal is to get agent $a$ to cell $(x - 1, y - 1)$, agent $a'$ to $(x - 1, y - 1)$, and WestGuide to $(x - 2, y - 1)$. First, Around($a'$, WestGuide) is executed. Then, agent $a$ finds the cell in distance one from $a'$ that is not occupied by WestGuide with LocalSearch(2) and enters the cell. Next, WestGuide finds the cell occupied by $a'$ with LocalSearch(1) and enters the corresponding cell. The routine is finished by performing Around(WestGuide, $a$) and Swap($a$, $a'$). See Figure 7 for an illustration.

**Turn Northwest at East Corner**  Assume that agents $a$, $a'$ and EastGuide reside in cells $(x + 1, y - 1)$, $(x, y - 1)$, and $(x, y)$, respectively. The goal is to get agent $a$ to cell $(x + 1, y)$ and agent $a'$ and WestGuide to $(x + 2, y - 1)$. First, agent $a$ uses LocalSearch(2) to find the cell $(x + 1, y)$, i.e., the cell adjacent to EastGuide that is not occupied by $a'$ and stores the corresponding port to its memory. Then, agent $a'$ enters cell $(x + 2, y - 1)$ by performing Around($a'$, $a$). Now, EastGuide can find its destination cell occupied by $a'$ by performing LocalSearch(3) and finally, agent $a$ goes to its destination cell by entering through the port it stored.
7.2 Triangle Search

Finally, we are able to specify our search protocol. We initialize the positions of the agents in the following manner. First, the NorthGuide moves to an adjacent cell, say the cell \((0, 1)\) from the origin. Note that the agents cannot distinguish between any of the cardinal directions. However, we can assume w.l.o.g. that the NorthGuide happens to step out of the origin towards north since the other cases work analogously.

Then, the WestGuide moves one step away from the origin to a cell not occupied by NorthGuide and performs LocalSearch(2) and if it finds a port sequence of length two that leads to NorthGuide without going through the origin, then WestGuide has occupied either cell \((1, 0)\) or \((-1, 0)\), say \((-1, 0)\). Otherwise, it repeats the initialization through another port from the origin. In an analogous fashion, EastGuide can find the cell \((1, 0)\).

Finally, \(a'\) performs Around\((a', \text{NorthGuide})\) assuming location \((0, 2)\) and agent \(a\) can reach the cell \((-1, 1)\) by finding the unique cell that is not the origin and has distance one to both agents NorthGuide and WestGuide.

Now, we can implement a protocol analogous to the one by Emek et. al. [16], where the agents visit every grid cell by moving according to triangles of increasing sizes. A triangle \(T_i\) of size \(i\) corresponds to the cells

\[
\{(j-i,j) \mid -i+1 \leq j \leq i\} \cup \{(j,-i+1) \mid -2i+1 \leq j \leq 2i-1\} \cup \{(j+i,-j) \mid -i \leq j \leq i-1\}.
\]

We split the triangle search into consecutive phases where one phase corresponds to exploring a triangle \(T_i\) for some \(i\). Refer to Figure 9 for an illustration.

The protocol for exploring triangle \(T_i\) is as follows: Assume that agents \(a, a', \text{NorthGuide}, \text{WestGuide}, \text{and EastGuide are located in cells } (-1, i-1), \ldots,
Note that the guides essentially mark a triangle that is one “smaller” than the triangle being explored. First, agents \(a\) and \(a'\) move towards southwest until the distance from both \(a\) and \(a'\) to WestGuide is exactly one. Note that given the initial locations, this configuration will be reached and agents \(a\) and \(a'\) can detect this configuration, e.g., by performing LocalSearch(2) after each movement step. Once the agents detect the above configuration, they perform turn east at southwest corner. Then, the agents move east until they reach a configuration where the distance from \(a'\) to EastGuide is exactly one. Now, they turn northwest at east corner and move northwest until the distance from \(a'\) to NorthGuide is exactly one. Turning at north corner finishes the exploration of \(T_i\).

We say that triangle \(T_i\) is explored correctly if the following properties hold

1. Agents \(a\), \(a'\), NorthGuide, WestGuide, and EastGuide are initially located in cells \((-1, i - 1), (0, i), (0, i - 1), (-2(i - 1) + 1, -i + 2),\) and \((2(i - 1) - 1, -i + 2)\) respectively.

2. After turning southwest at the end of exploring \(T_i\), agents \(a\), \(a'\), NorthGuide, WestGuide, and EastGuide are located in cells \((-1, i), (0, i + 1), (0, i), (-2i + 1, -i + 1),\) and \((2i - 1, -i + 1)\), respectively.

3. Every cell in \(T_i\) is visited by at least one agent.

Now, it is easy to verify the following observation that leads to Theorem 6 by observing that \(T_i\), for any \(i\), takes finite time.

Observation 5. Every triangle \(T_i\) is explored correctly.

**Theorem 6.** Five asynchronous agents can find the treasure in the port numbering model.

8 Conclusion

In this paper, we considered the collaborative grid exploration problem with agents controlled by asynchronous finite automata. While this paper shows tight bounds for the minimum number of agents required to explore the grid in different timing models, it remains an intriguing open question to generalize these results, especially finding non-trivial lower bounds, to more complex structures.

References

[1] Susanne Albers and Monika Henzinger. Exploring Unknown Environments. *SIAM Journal on Computing*, 29:1164–1188, 2000.

[2] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovasz, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *FOCS*, pages 218–223, 1979.

[3] Igor Averbakh and Oded Berman. A heuristic with worst-case analysis for minimax routing of two travelling salesmen on a tree. *Discrete Appl. Math.*, 68(1-2):17–32, 1996.
[4] Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the Plane. *Information and Computation*, 106:234–252, 1993.

[5] M. A. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *FOCS*, pages 75–85, 1994.

[6] M. Blum and W. J. Sakoda. On the capability of finite automata in 2 and 3 dimensional space. In *FOCS*, pages 147–161, 1977.

[7] Manuel Blum and Dexter Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *FOCS*, pages 132–142, 1978.

[8] Lothar Budach. Automata and labyrinths. *Mathematische Nachrichten*, 86(1):195–282, 1978.

[9] Marek Chrobak, Leszek Gasieniec, Thomas Gorry, and Russell Martin. *Group Search on the Line*, pages 164–176. Springer Berlin Heidelberg, 2015.

[10] Lihi Cohen, Yuval Emek, Oren Louidor, and Jara Uitto. Exploring an infinite space with finite memory scouts. In *SODA*, pages 207–224, 2017.

[11] Xiaotie Deng and Christos Papadimitriou. Exploring an Unknown Graph. *Journal of Graph Theory*, 32:265–297, 1999.

[12] Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc. Tree Exploration with Little Memory. *Journal of Algorithms*, 51:38–63, 2004.

[13] Yann Disser, Jan Hackfeld, and Max Klimm. Undirected graph exploration with $\theta(\log \log n)$ pebbles. In *SODA*, pages 25–39, 2016.

[14] Stefan Dobrev, Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. *Finding Short Right-Hand-on-the-Wall Walks in Graphs*, pages 127–139. Springer Berlin Heidelberg, 2005.

[15] Christian A. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. *ACM Trans. Algorithms*, 2(3):380–402, 2006.

[16] Yuval Emek, Tobias Langner, David Stolz, Jara Uitto, and Roger Wattenhofer. How many ants does it take to find the food? *Theor. Comput. Sci.*, 608:255–267, 2015.

[17] Yuval Emek, Tobias Langner, Jara Uitto, and Roger Wattenhofer. Solving the ANTS Problem with Asynchronous Finite State Machines. In *ICALP*, pages 471–482, 2014.

[18] Ofer Feinerman, Amos Korman, Zvi Lotker, and Jean-Sebastien Sereni. Collaborative Search on the Plane Without Communication. In *PODC*, pages 77–86, 2012.

[19] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed coordination of a set of autonomous mobile robots. In *Intelligent Vehicles Symposium*, pages 480–485, 2000.

[20] Pierre Fraigniaud and David Ilcinkas. *Digraphs Exploration with Little Memory*, pages 246–257. Springer Berlin Heidelberg, 2004.
[21] Frank Hoffmann. One pebble does not suffice to search plane labyrinths. In
FCT, pages 433–444, 1981.

[22] Alejandro López-Ortiz and Graeme Sweet. Parallel Searching on a Lattice.
In CCCG, pages 125–128, 2001.

[23] Petrișor Panaite and Andrzej Pelc. Exploring unknown undirected graphs.
In SODA, pages 316–322, 1998.

[24] H. A. Rollik. Automaten in planaren graphen, pages 266–275. Springer
Berlin Heidelberg, Berlin, Heidelberg, 1979.

[25] Kazuo Sugihara and Ichiro Suzuki. Distributed algorithms for formation of
geometric patterns with many mobile robots. Journal of Robotic Systems,
13(3):127–139, 1996.

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

[27] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile
robots - formation and agreement problems. In SIROCCO, pages 1347–1363,
1996.