UNIVERSITY OF THESSALY

DOCTORAL THESIS

Solving Fundamental Problems in Hostile Networks

Author:

Nikos GIACHOUDIS

Supervisor:

Dr. Euripides MARKOU

A thesis submitted in fulfillment of the requirements

for the degree of Doctor of Philosophy

in the department of

Computer Science and Biomedical Informatics

August 4, 2021

UNIVERSITY OF THESSALY

School of Science

Computer Science and Biomedical Informatics

Solving Fundamental Problems in Hostile Networks

PhD Thesis

of

Nikos Giachoudis

Advising Commitee: Euripides Markou Vassilis Plagianakos Aris Pagourtzis

Assessment Committee

Euripides Markou	University of Thessally, Greece
Vassilis Plagianakos	University of Thessally, Greece
Aris Pagourtzis	National Technical University of Athens, Greece
Evangelos Bampas	Université Paris-Saclay, France
Thanasis Loukopoulos	University of Thessally, Greece
Orestis Telelis	University of Piraeus, Greece
Theodoros Tzouramanis	University of Thessally, Greece

Lamia, August 4, 2021

Abstract

Nikos GIACHOUDIS

Solving Fundamental Problems in Hostile Networks

In this thesis we study three fundamental problems in hostile network environments. Our work lies in the area of Theoretical Computer Science and in particular in the area of Distributed Computing. We design distributed algorithms and we prove their correctness and computational complexity for the following problems. We study how identical mobile agents that move autonomously in a network and have only local information about the environment, can gather at a node of a network despite the presence of a hostile mobile agent which may prevent agents from visiting a node of the network. We also study how a number of agents that move autonomously and have only local information about the environment, might collectively decontaminate a network in which there is a virus capable of infecting an area of the network. Finally, we study how mobile agents that move autonomously in a network can learn a message initially carried by only one of the agents despite the fact that the agents cannot communicate from a distance, cannot leave messages at nodes of the network and many links of the network might unpredictably fail at any time. The above problems are often needed to be solved as an early step in order to solve other problems in distributed environments. Some of those problems have been previously studied either in safe environments or when static hostile entities are present. Here we investigate more general models in which either hostile mobile agents exist in the network, or a hostile behaviour can spread at the nodes of the network, or many unpredictable faults can occur in the network. We prove lower bounds on the number of mobile agents which are necessary in order to solve the problems and design optimal algorithms that meet those lower bounds. We show how

those three important problems can be solved fast in severe hostile environments by autonomous mobile agents having very limited resources (e.g., very limited memory and no initial knowledge about the environment) and capabilities (e.g., no communication from a distance and no sensory abilities). Since in most cases our models are very weak, we hope that our techniques can be extended to handle different problems and systematically increase our knowledge in extremely malicious environments where the hostile behaviour can affect large and sometimes distant areas of the environment.

Περίληψη

Νίκος Γιαχούδης

Αλγοριθμική Αντιμετώπιση Προβλημάτων σε μή-Ασφαλή Δίκτυα

Σε αυτή τη διατριβή μελετάμε τρία βασικά προβλήματα σε περιβάλλοντα εχθρικών δικτύων. Το έργο μας βρίσκεται στον τομέα της Θεωρητικής Πληροφορικής και ειδικότερα στον τομέα των Κατανεμημένων Υπολογισμών. Σχεδιάζουμε κατανεμημένους αλγόριθμους και αποδεικνύουμε την ορθότητα και την υπολογιστική πολυπλοκότητά τους για τα ακόλουθα προβλήματα. Μελετάμε πώς πανομοιότυποι πράκτορες που κινούνται αυτόνομα σε ένα δίκτυο και έχουν μόνο τοπικές πληροφορίες για το περιβάλλον, μπορούν να συγκεντρωθούν σε έναν κόμβο ενός δικτύου παρά την παρουσία ενός εχθρικού κινητού πράκτορα που μπορεί να εμποδίσει τους πράκτορες να επισκεφθούν έναν κόμβο του δικτύου. Μελετάμε επίσης πώς ένας αριθμός πρακτόρων που κινούνται αυτόνομα και έχουν μόνο τοπικές πληροφορίες σχετικά με το περιβάλλον, ενδέχεται να καθαρίσουν συλλογικά ένα δίκτυο στο οποίο υπάρχει ένας ιός ικανός να μολύνει μια περιοχή του δικτύου. Τέλος, μελετάμε πώς πράκτορες που κινούνται αυτόνομα σε ένα δίκτυο μπορούν να μάθουν ένα μήνυμα που αρχικά το γνωρίζει μόνο ένας πράκτορας, παρά το γεγονός ότι οι πράκτορες δεν μπορούν να επικοινωνήσουν από απόσταση, δεν μπορούν να αφήσουν μηνύματα σε κόμβους του δικτύου και η οποιαδήποτε ακμή του δικτύου ενδέχεται να αποτύχει απρόβλεπτα ανά πάσα στιγμή. Τα παραπάνω προβλήματα συχνά χρειάζεται να επιλυθούν ως πρώιμο βήμα για την επίλυση άλλων προβλημάτων σε κατανεμημένα περιβάλλοντα. Μερικά από αυτά τα προβλήματα έχουν μελετηθεί προηγουμένως είτε σε ασφαλή περιβάλλοντα ή όταν υπάρχουν στατικές εχθρικές οντότητες. Εδώ διερευνούμε πιο γενικά μοντέλα στα οποία είτε υπάρχουν στο δίκτυο εχθρικοί κινητοί πράκτορες, είτε μια εχθρική συμπεριφορά μπορεί να εξαπλωθεί στους κόμβους του δικτύου ή πολλά

απρόβλεπτα σφάλματα μπορούν να προκύψουν στο δίκτυο. Αποδεικνύουμε κάτω φράγματα στον αριθμό των κινητών πρακτόρων που είναι απαραίτητοι για την επίλυση των προβλημάτων και σχεδιάζουμε βέλτιστους αλγόριθμους που ικανοποιούν τα κάτω φράγματα. Δείχνουμε πώς αυτά τα τρία σημαντικά προβλήματα μπορούν να επιλυθούν αποδοτικά σε πολύ εχθρικά περιβάλλοντα από αυτόνομους κινητούς πράκτορες που έχουν πολύ περιορισμένους πόρους (π.χ. πολύ λίγη μνήμη και χωρίς αρχικές γνώσεις για το περιβάλλον) και δυνατότητες (π.χ., χωρίς επικοινωνία από απόσταση και χωρίς αισθητηριακές ικανότητες). Επειδή στις περισσότερες περιπτώσεις τα μοντέλα μας είναι πολύ αδύναμα, ελπίζουμε ότι οι τεχνικές μας μπορούν να επεκταθούν για να χειριστούν διαφορετικά προβλήματα και να αυξήσουν συστηματικά τις γνώσεις μας σε εξαιρετικά κακόβουλα περιβάλλοντα όπου η εχθρική συμπεριφορά μπορεί να επηρεάσει μεγάλες και μερικές φορές απομακρυσμένες περιοχές του περιβάλλοντος.

Acknowledgements

I wish to express my deepest gratitude to my advisor Euripides Markou. He was there at all times answering every question I might had and passing his knowledge through his own unique way. The most things I learned, I learned them because of Euripides and I would be lost if he wasn't there to guide me. He is a very passionate scientist and with his guidance I was able to achieve the feat of this thesis.

I also wish to show my gratitude to the rest of the advisory committee, Vassilis Plagianakos and Aris Pagourtzis, who were ready to help me in any case, and the rest of the examination committee, Thanasis Loukopoulos, Evangelos Bampas, Orestis Telelis, and Theodoros Tzouramanis, for their feedback which is one of the most important steps in order to have a good result. It is whole-heartedly appreciated that their great advice for my study proved monumental towards the success of this thesis. I wish to thank all of my professors who showed me what a beautiful world science is, and all of my colleagues with which I could talk and exchange thoughts in order to keep our research going.

Finally, I am indebted to all of my family and friends who were there and had to listen to all those things that I wanted to tell about my research which was very hard for them to understand, but in the process of explaining them it helped me having a better understanding too. My parents and my sister that supported me all of those years that I was trying quite hard to accomplice such an important thing for me. But most of all I would like to pay my special regards to my wife Angeliki, who came into my life during my PhD and with patience made me better and supported me even if it was one of the most difficult things to do.

Nikos Giachoudis

August 4, 2021

Contents

List of Figures

To Angeliki

1 Introduction

We often need to design algorithms used by entities which act autonomously having very little or no initial knowledge about the environment, very limited or no communication or sensor capabilities. To answer whether a problem can be collectively solved by such weak entities possibly tolerating faults or some times acting in hostile environments is not important and interesting only for theoriticians but also in practise. The challenges behind designing optimal distributed algorithms for such weak models and prove their correctness as well as showing in what models certain problems are infeasible and carefully adding the necessary capabilities that makes a problem feasible, make the issue very attractive in Theoretical Computer Science. However, such weak models are often very useful in practice, not only because many real problems are needed to be solved by entities that act autonomously in hostile environments, but also because it is usually easier to implement weaker models and at a lower cost.

For example, suppose that a number of robots are scattered in an environment, and they need to gather at the same point. Each robot has only limited information about the total number of robots and the initial configuration. It can sense the environment up to a fixed radius, it can only communicate with close-by robots and behaves autonomously. Moreover, the robots might have different coordinate systems and might not be synchronized. Furthermore, some of the robots might temporarily shut down. In other scenarios the robots might operate in a network being able to move using the links of the network. Hostile entities might exist in the network capable of damaging the robots, or at least preventing them from effectively performing a task. The links of the network may fail from time to time while the robots might need to discover and report hostile activities or might need to solve a problem like broadcasting, despite hostile activities. Alternatively, the nodes of the network might represent hosts and instead of robots, a piece of information and/or software might be able to replicate itself and move from host to host

using the links of the network, in order to maintain the network.

Such problems arise very often due to the heavy use of networks but also because of the need to handle scenarios and problems where the structures are huge or it is just not desirable to have a central authority that instructs all entities how to act. Hence it is often more realistic (and applicable) to design *distributed* algorithms that can be used autonomously by mobile entities so that the problem can be solved despite the absence of a central authority and/or in a hostile environment.

Although there are many interesting results on what can be achieved in such scenarios even in hostile environments, most of the work either involves non-faulty environments or environments with static hostile entities. Our knowledge on what can be achieved when the hostile entities are able to move or many faults can arise in distant areas of the environment, is rather limited. In this thesis we study three problems in such hostile environments.

We study how identical mobile agents that move autonomously in a network and have only local information about the environment, can gather at a node of a network despite the presence of a hostile mobile agent which may prevent agents from visiting a node of the network ([34]). We also study how a number of agents that move autonomously and have only local information about the environment, might collectively decontaminate a network in which there is a virus capable of infecting an area of the network ([56]). Finally, we study how mobile agents that move autonomously in a network can learn a message initially carried by only one of the agents despite the fact that the agents cannot communicate from a distance, cannot leave messages at nodes of the network and many links of the network might unpredictably fail at any time ([37]). The above problems are often needed to be solved as an early step in order to solve other problems in distributed environments. Some of those problems have been previously studied either in safe environments or when static hostile entities are present. Here we investigate more general models in which either hostile mobile agents exist in the network, or a hostile behaviour can spread at the nodes of the network, or many unpredictable faults can occur in the network. We prove lower bounds on the number of mobile agents which are necessary in order to solve the problems and design optimal algorithms that meet those

lower bounds. We show how those three important problems can be solved fast in severe hostile environments by autonomous mobile agents having very limited resources (e.g., very limited memory and no initial knowledge about the environment) and capabilities (e.g., no communication from a distance and no sensory abilities). Since in most cases our models are very weak, we hope that our techniques can be extended to handle different problems and systematically increase our knowledge in extremely malicious environments where the hostile behaviour can affect large and sometimes distant areas of the environment.

This thesis is divided into five chapters. Chapter 1 is a quick introduction into the field of distributed algorithms and a summary of related work that has been done for the problems studied here, as well as other similar problems. Chapter 2 consists of definitions and notations that are needed in order to understand the rest of the chapters. Finally, Chapters 3-5 give results on fundamental problems in a distributed context. Specifically, the results concern three problems, as mentioned above, and each problem is studied separately in each corresponding chapter.

1.1 Related work

There is an extended research on the problems studied in this thesis and briefly mentioned above. We will discuss the previously known results on those problems and also on related problems.

Let us discuss about the gathering problem. The problem is called *rendezvous* when the mobile agents are only two, *gathering* otherwise [65], and has been widely studied when the environment is modelled as a graph and the mobile agents can move along the edges of the graph. However, most of the studies are restricted to fault-free environments and little is known about gathering when the environment is faulty or hostile. Possible faults can be a permanent failure of a node, like for example the so called *black hole* that destroys agents arriving at a node [6, 40], or, transient faults that can appear anywhere in the graph and are controlled by a mobile hostile entity (an *intruder*) that behaves maliciously [8].

Most of the research that has been done in hostile environments is on the direction of how to locate a malicious node in a graph (see, e.g., [41, 61]). Protecting the network against a malicious entity able to move along the edges of the graph, is generally a more difficult problem. Problems in this direction include the so-called *network decontamination* or *intruder capture* problem (see, e.g., [47, 67]). Other types of faults or malicious behavior that have been considered in the context of the gathering problem are *Byzantine agents* [23, 27, 39], and *delay-faults* [20], and *edge evoling graphs* [79].

A Byzantine agent is indistinguishable from the legitimate or *honest* agents, but it may behave in an arbitrary manner and may provide false information to the honest agents in order to induce them to make mistakes, thus preventing the rendezvous of the honest agents. However the Byzantine agents cannot actively harm the honest agents or physically prevent the agents from moving. Delay-faults [20] can prevent an agent from moving for an arbitrary but finite time (i.e., they must eventually allow the agent to move), whereas probabilistic edge evolving graphs are dynamic networks where the set of communication links continuously changes thus preventing the use of standard gathering algorithms that work for static networks [79]. Finally, the gathering problem has been also studied in the plane when there are faulty agents which may crash [2, 13].

We consider here a relatively new type of *malicious agent* that was first introduced in [35] and successively investigated in [32, 33]. This malicious agent can move arbitrarily fast along the edges of a graph, it has full information about the graph and the location of all other honest agents, and it even has full knowledge of the actions that will be taken by the agents. The objective of the malicious agent is to prevent the honest agents from gathering by blocking their path. More precisely, when the malicious agent occupies a node u of the graph, it can prevent (or *block*) the movement of any *honest* agent to node u, and at the same time is detected by those agents. In [32, 33, 35] it is shown how this malicious agent is a stronger adversary than the Byzantine agent or the Intruder agent, or the delay faults, as even one malicious agent can prevent rendezvous of honest agents in many cases.

The gathering problem has been investigated in [33, 35] for *asynchonous* mobile agents moving in a ring or in a grid with one malicious agent. The asynchrony among the honest agents combined with the ability of the malicious agent to move arbitrarily fast, gave the

power to the malicious agent to prevent the next move of several agents at any stage of the algorithm. Thus, gathering of the honest agents was shown to be impossible [35], in all cases except when the agents started from an initial configuration where the subgraph induced by all the occupied nodes was connected, and additionally, in the case of the grid graph, when the honest agents were able to "see" nodes at distance two in order to check if they were occupied or not.

To allow the possibility of gathering the honest agents in more scenarios, we relax the constraint of asynchrony and assume that the honest agents move in synchronous steps. Note that, if two synchronous honest agents try to move to two distinct nodes at the same time, then the malicious agent can block the move of at most one agent, even though the malicious agent can execute moves of arbitrary distances between two consecutive steps of the honest agents. This model with synchronous agents was studied in [32, 33] only for the case of a ring, where it was already possible to solve gathering in more cases than with asynchronous agents.

The problem of Black Virus Decontamination (BVD) is a well known problem in the literature too. It combines the Black Hole Search (BHS) problem and the Intruder Capture (IC) problem. The Black Hole Search problem has been extensively studied for various topologies and communication models (e.g., in [24, 28, 42, 43]). A recent survey of the results on the black hole search problem can be found in [70]. The Intruder Capture problem has been also studied (e.g., in [10, 12, 38, 48, 51, 71]). The BVD problem was introduced in [16] where the problem was studied in specific topologies, namely *q-grids*, *q-tori* and *hypercubes*. The problem was considered for a team of initially co-located mobile agents that are injected somewhere in the network and a different solution protocol was given for each topology. In [17] the BVD problem was studied in arbitrary networks, where the agents have a map of the network and can compute an optimal exploration sequence. Furthermore, in [17] two types of black virus clones are considered; *fertile clones*, that maintain the same capabilities as the original black virus and *sterile clones* that cannot spread when visited by an agent. In [18] a protocol providing a distributed optimal solution for arbitrary graphs is presented. The main difference between [17] and [18] is that in [18] the agents have '2-hop visibility' instead of a map. Moreover, only the case of *sterile clones* is considered in [18]. In [15] the problem was considered for

arbitrary networks that contain multiple black viruses for the first time. The agents in [15] are provided with '2-hop visibility' and both the cases of *sterile* and *fertile* clones are investigated. Finally, the BVD problem has been studied in chordal rings in [3] and parallel strategies for the decontamination were given in [66].

The problem of broadcasting has been originally investigated in message passing multi-hop radio networks (e.g. see the survey [53]). Previous studies on broadcast and other communication problems have focused on the efficiency of performing the task, either in terms of time taken [22], or in terms of energy expended [26]. A slightly different line of research considers the problem of communicating in the presence of faults and the objective is to tolerate as many faults as possible. The faults can be missing links or nodes [54] in the network or loss of messages [55], in case of message passing networks. Recently there has been a lot of interest in so called *dynamic networks* which model both faults and changes in network topology in a uniform manner by considering that the network may change in each round during the execution of the algorithm. The *evolving graph* model [44] represents a dynamic network by a sequence of graphs $\mathcal{G} = G_1, G_2, \ldots$ based on the same set of vertices V but the set of edges changes in each round i, i.e. each graph $G_i = (V, E_i)$ is a spanning subgraph of the graph $G = (V, \cup E_i)$, which is called the footprint of the dynamic network. For solving most problems, some assumptions about the connectivity of the dynamic network need to be made. We consider the model of *constantly connected* dynamic networks which is discussed in detail in Section 2.4.3. Note that, these assumptions are much weaker compared to that of T-interval connected networks as in several previous results [60, 64] where the network is assumed to contain a stable spanning tree for a continuous period of T rounds.

2 Preliminaries

In this chapter we discuss some preliminary definitions and observations that are going to be used in the next chapters.

The environment where, as we will see later the entities operate in, is also the environment where the problems are defined. There are two main categories for the environment where related problems have been studied: a discrete environment and a continuous environment [49]. The continuous environment is usually defined as a space in $\mathbb{R}^d$, where d is the dimension of that space, and the discrete environment is usually defined as a graph.

2.1 Network

Definitions of models of a network come from the algorithmic theory of distributed computing (see for example [63, 68, 72]). Following the definition in [49], the network is usually defined or represented as a simple, finite, connected undirected graph $G = (V, E)$, where V is the set of nodes and $E \subseteq V \times V$ is a set of edges (also mentioned as links). The nodes of the network may or may not have distinct identities. Often the network is *anonymous* (see [49, 63]), i.e., the nodes do not have any labels. Edges may or may not have labels. For example, as defined in [49] the incident links at each node are labeled as follows. Let $V(u) = \{v \in V : (u, v) \in E\}$ denote the neighbours of u, $E(u)$ denote the set of edges incident to node u and $\lambda_u : E(u) \to \mathcal{L}$ be an injective function which associates a distinct label (also mentioned as *port number*) from a set of labels $\mathcal{L}$ to each incident edge of u. Hence, for each edge $e = (u, v)$ there are two associated labels, namely $\lambda_u(e)$ and $\lambda_v(e)$, which may be different (for example see Figure 2.1). The labeling of graph G is a set $\lambda = \{\lambda_u : u \in V\}$ and the pair (G, λ) is the corresponding edge-labeled graph, also sometimes called *graph world* or *netscape*.

Another characteristic used for each link is a *queuing policy* [72]. An edge (or link) (u, v) represents a channel of communication between u and v and when a message is sent from one node to the other, the message actually gets transmitted via this channel. We can have different outcomes when two or more messages are transmitted at the same time via a channel. The ordering of the delivery of the messages may or may not be the same as the ordering of transmission. The specific behaviour for all the edges of the network is defined in the model for each problem. An example of a link behaviour is that the link follows the principles of a queue. Queues follow a first-in-first-out (FIFO) principle and this is the behaviour of moving in such a link, i.e., the order of reception is the same as the order of transmission.

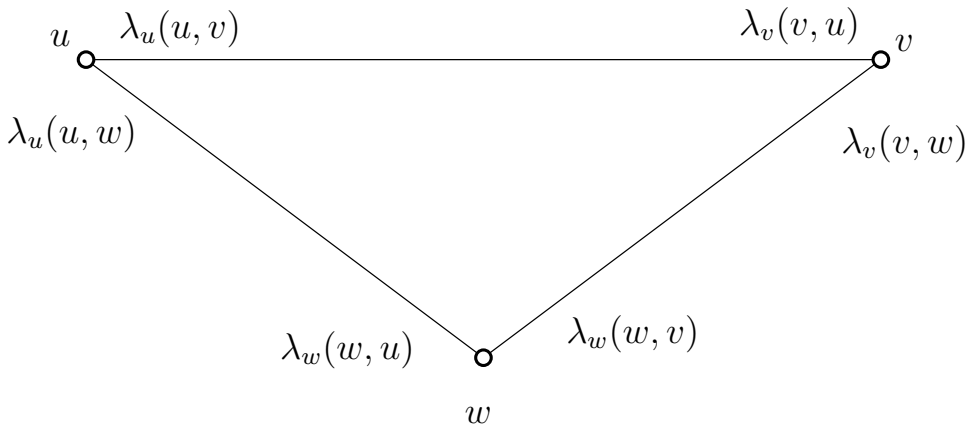

FIGURE 2.1: Example of an edge labeling.

2.1.1 Network topologies

Usually problems are studied in specific network *topologies* [49, 63, 68, 72]. A topology is the basic structure of the network and it is represented by classical graph families [74]. We provide some definitions and give examples for some graph families we use in our work.

Definition 2.1 (*cyclic graph* [74])**.** *If $n \in \mathbb{Z}$ and $n \geq 3$, the cyclic graph on n vertices, denoted C_n, is the simple graph having vertex set $\{1, 2, 3, \ldots, n\}$ and edge set $\{\{1, 2\}, \{2, 3\}, \ldots, \{(n-1), n\}, \{n, 1\}\}$ (see Figure 2.2 a).*

Definition 2.2 (*complete graph* [74])**.** *If $n \in \mathbb{Z}^+$, the complete graph on n vertices, denoted K_n, is the simple graph having vertex set $\{1, 2, 3, \ldots, n\}$ and all possible edges.*

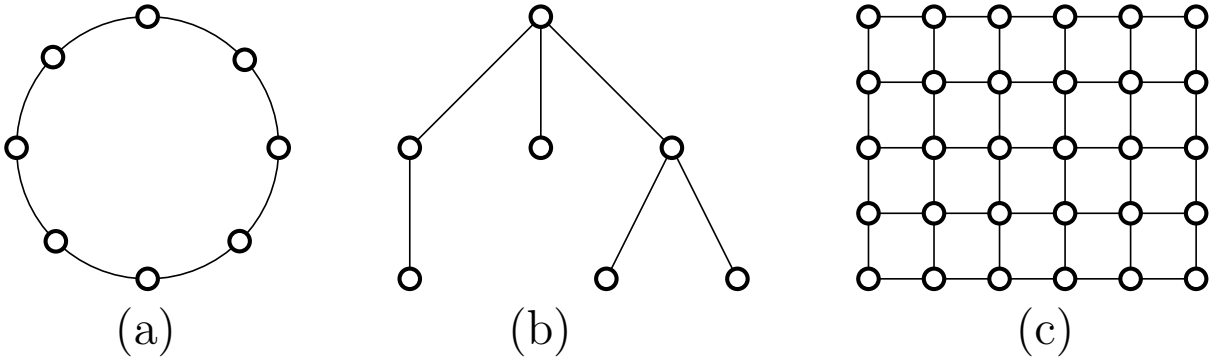

FIGURE 2.2: Example of graph topologies.

Definition 2.3 (*tree* [78]). *If $n \in \mathbb{Z}^+$, a tree graph on n vertices, denoted T_n, is the simple graph that is connected and acyclic (see Figure 2.2 b).*

Definition 2.4 (*star* [78]). *If $n \in \mathbb{Z}^+$, the star graph on n vertices, denoted S_n, is the graph having vertex set $\{1, 2, 3, \ldots, n\}$ and edge set $\{\{1, 2\}, \{1, 3\}, \ldots, \{1, n\}\}$.*

Definition 2.5 (*line graph* [78]). *If $n \in \mathbb{Z}^+$, the line graph on n vertices, denoted P_n, is the simple graph having vertex set $\{1, 2, 3, \ldots, n\}$ and edge set $\{\{1, 2\}, \{2, 3\}, \ldots, \{(n-1), n\}\}$ (see Figure 2.4 d).*

Definition 2.6 (*cactus* [37]). *A cactus graph is a connected simple graph in which any two simple cycles have at most one vertex in common (see Figure 2.3.*

Definition 2.7 (*hypercube* [58]). *A hypercube graph Q_n is the simple graph formed from the vertices and edges of an n-dimensional hypercube (see Figure 2.4 e, for a 3-dimensional hypercube).*

Definition 2.8 (*grid* [75]). *A two-dimensional grid graph, also known as a rectangular grid graph or two-dimensional lattice graph [1], is an $n \times m$ lattice graph that is the graph Cartesian product $P_n \times P_m$ of line graphs on n and m vertices. The $n \times m$ grid graph is sometimes denoted $L(n, m)$ [1] (see Figure 2.2 c for a 5×6 grid graph)*[1]

2.1.2 Sense of direction

In a labeled graph (G, λ) a *sense of direction* can be defined [45,46,49,50,63]. Let π denote a path in G and let $P[x]$ denote the set of all the non empty paths that start from $x \in V$. Furthermore, $P[x, y]$ is the set of all different paths that start from $x \in V$ ending to $y \in V$,

[1] In chapters 3 and 5 the origin of the grid, denoted by $(0, 0)$, is defined as needed for each case.

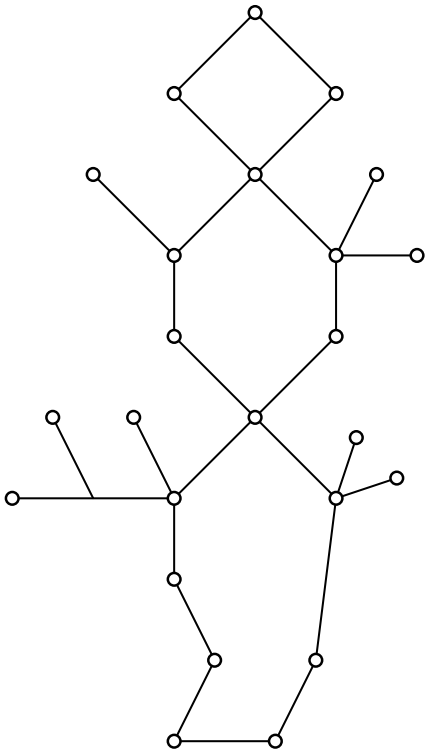

FIGURE 2.3: Example of cactus graph topology.

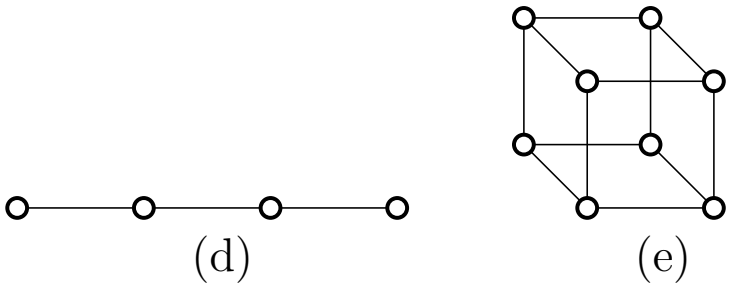

FIGURE 2.4: Example of graph topologies.

$y \neq x$. Let $\Lambda_x : P[x] \to \mathcal{L}^+$ and $\Lambda = \{\Lambda_x : x \in V\}$ be the extensions of λ_x and λ for paths respectively. Finally, let $\Lambda[x] = \{\Lambda_x(\pi) : \pi \in P[x]\}$ and $\Lambda[x, y] = \{\Lambda_x(\pi) : \pi \in P[x, y]\}$.

Informally, we say that an edge-labeled graph (G, λ) has a sense of direction when there is a consistent coding and decoding function f, which maps paths to values. Following the description in [46], given (G, λ), a *consistent coding function* f for λ is a function, which has $\mathcal{L}^+$ as domain, and maps paths starting from the same node to the same value if and only if they end at the same node. More formally, $\forall x, y, z \in V$, and $\forall \pi_1 \in P[x, y]$ and $\forall \pi_2 \in P[x, z]$, $f(\Lambda_x(\pi_1)) = f(\Lambda_x(\pi_2)) \Leftrightarrow y = z$.

Definition 2.9 (*Weak sense of direction* [45])**.** *A system (G, λ) has a weak sense of direction iff there exists a coding function f for λ.*

Let us denote the codomain of f as $\mathcal{N}$. A decoding function h for f is a function $h : \mathcal{L} \times \mathcal{N} \to \mathcal{N}$ such that $\forall x, y, z \in V$ and $\langle x, y\rangle \in E(x)$, where $E(x)$ is the incident edges to x, and $\pi \in P[y, z] : h(\lambda_x(\langle x, y\rangle), f(\Lambda_y(\pi))) = f(\lambda_x(\langle x, y\rangle) \cdot \Lambda_y(\pi))$, where $\cdot$ is the concatenation operator. Informally, a decoding function maps the pair of an edge label $\lambda_x(\langle x, y\rangle)$ and an encoding of a path to the same value as the encoding of the path starting from edge $\langle x, y\rangle$.

Definition 2.10 (Sense of direction [45])**.** *A system (G, λ), has a sense of direction iff the following conditions hold:*

1. *there exists a coding function f for λ,*
2. *there exists a decoding function h for f.*

We say a graph is oriented when it has a sense of direction. For example, an *oriented grid graph* is a grid graph where its nodes are labeled consistently with the four labels, north, south, west, and east (see Figure 2.5).

2.1.3 Time-varying graphs

Following the description in [49], we mention that there are many models of graphs that change over time like time-varying graphs, evolving graphs, temporal graphs etc. As mentioned in [49] time-varying graphs are the most general ones that cover most cases. The definition of time-varying graphs follows.

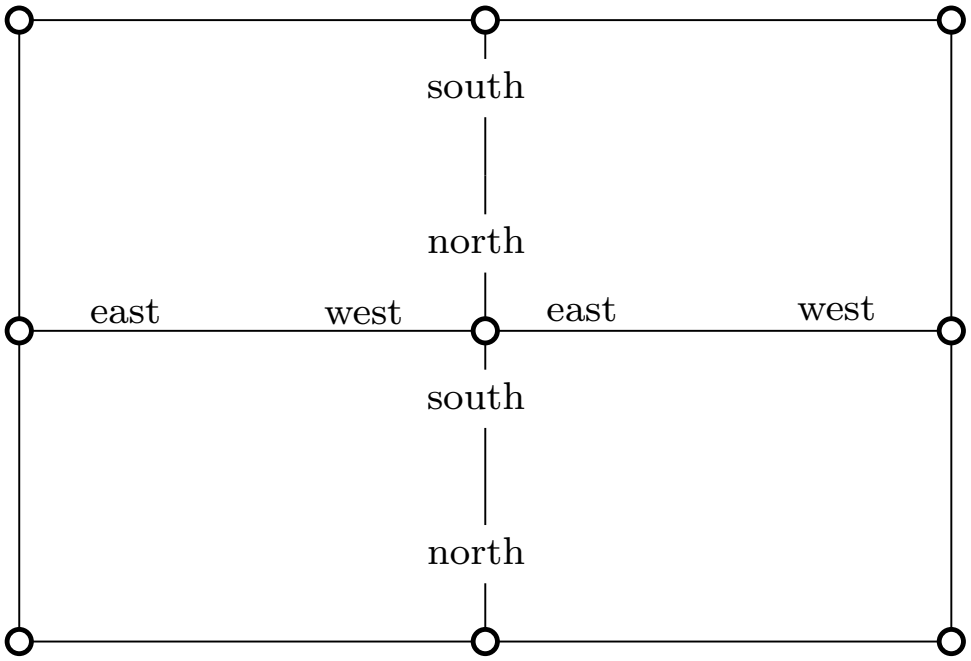

FIGURE 2.5: An example of an oriented grid-graph. Notice that the port labeling is globally consistent.

A *time-varying graph* [19] or TVG for short, is a graph that changes over time. It is defined as a quintuple $\mathcal{G} = (V, E, \mathcal{T}, \zeta, \rho)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. $\mathcal{T} \subseteq \mathbb{T}$ is the time-span of the time, also mentioned as lifetime of the system, where $\mathbb{T}$ is the temporal domain which is $\mathbb{N}$ for discrete systems and $\mathbb{R}^+$ for continuous time systems. Lastly, ρ and ζ are two functions. Specifically, $\rho : E \times \mathcal{T} \to \{0, 1\}$ is the function of edge presence. It shows if a specific edge is present at a specific time or not. Function $\zeta : E \times \mathcal{T} \to \mathbb{T}$ is a latency function that shows how much time is needed for an agent to cross a specific edge starting at a specific time. The static graph which consists of all the edges available during the lifetime of $\mathcal{G}$ is called the footprint of $\mathcal{G}$.

As stated in [49] dynamic networks are usually considered in a synchronous or semi-synchronous models. In those models time is discrete (i.e., $\mathbb{T} = \mathbb{N}$ and is measured in rounds, and the latency function is constant $\zeta = 1$, meaning that an agent needs one time unit to traverse an edge. With these assumptions, the TVG is commonly viewed as a series of static graphs $G_1, G_2, \ldots$ and the network that changes over time is called an *evolving graph*. Specifically, G_t is the static snapshot of $\mathcal{G}$ at time t [44, 57].

2.2 Mobile agents

Based on the description of [69], a *mobile agent* is modeled by a finite state machine (initially that definition was used by Shannon in 1951 [73]). Formally, a mobile agent is defined as a hexad $\mathcal{A} = (X, Y, \mathcal{S}, \delta, \lambda, S_0)$, where $X \subseteq \mathcal{D}_v \times \mathcal{C}_v, Y \subseteq \mathcal{D}_v \times \{actions\}$. $\mathcal{S}$ is a set of states that a mobile agent can be in, with $S_0 \in \mathcal{S}$ being the *initial* state. $\delta : \mathcal{S} \times X \to \mathcal{S}$ is the state transition function of the mobile agent and $\lambda : \mathcal{S} \to Y$ is a function that takes a state of the mobile agent and gives the move the mobile agent will make. $\mathcal{D}_v$ is the set of edge labels available to the mobile agent that occupies node v and $\mathcal{C}_v$ is the state of node v (for example if there are any other agents on that node, if there is some kind of information on that node etc).

Mobile agents move in the following way. An agent starts in state S_0 and from that its first move, indicated by $\lambda(S_0)$, is decided. When the agent makes a move from node u to node v it executes the following tasks: it reads the port label $i \in \mathcal{D}_v$ it came from and the state $c_v \in \mathcal{C}_v$ of v and with the pair (i, c_v) decides its new state $S' = \delta(S, (i, c_v))$ using the state transition function, where S is the previous state the agent was in. The new state S' indicates the new move $\lambda(S')$ of the agent. The mobile agent continues moving in the network like this, possibly for ever.

2.2.1 Communication

As mentioned in [49], the most crucial part of distributed computing with mobile agents is the communication between them, or as referred to in [14] *coordination*. Following the same classification as in [49] the different coordination models depend on three basic characteristics, temporal and spatial requirements, as well as in the explicitness of the information that is being communicated. We summarize the combination of those characteristics with the examples of models in Table 2.1.

Let us now discuss each communication model separately. Following the discussion in [49], the most powerful model is the $\mathcal{G}$LOBALSPACE. In this model the mobile agents share a common space called *globalspace* in order to communicate. The operations provided are exclusive-write and concurrent-read that can be used from any node on the network. This model is also referred as *associative blackboard* [14]. The access to the

Model	Spatial requirement	Temporal requirement
globalspace		
whiteboard	✓	
wireless		✓
face-to-face	✓	✓
tokens (implicit)	✓	
beeping (implicit)		✓

TABLE 2.1: Communication models [14, 49]

globalspace has neither spatial nor temporal constraints and the information shared between the mobile agents is explicit. To be clear in this model the only way of communicating is through the globalspace, and the mobile agents cannot even distinguish if there are other agents on the same node.

The opposite side is the $\mathcal{F}$ACE-TO-$\mathcal{F}$ACE model. This is the simplest [49] model and there are both spatial and temporal constraints. In order for the mobile agents to communicate under the face-to-face model they have to be on the same node v of the network at the same time. In such a case each agent can read the memory and/or state of the other colocated agents and use that information to decide its next move.

An extensively studied model is the $\mathcal{W}$HITEBOARD model [4, 9, 29, 30, 49], which is spatially restricted but not temporally restricted. In this model there is a "whiteboard" (i.e. a local shared memory) at each node $v \in V$ of the network, which the mobile agents can access with the restriction of fair mutual exclusion in order to give access to at most one agent at the same time, and any agent will be given access to the whiteboard within a finite amount of time. The mobile agents that are at a node $v \in V$ of the network can read and write explicit information on the whiteboard of v and use that information to decide their next move. In the case of *anonymous* agents, the author of any information on the whiteboard is unknown and the size of the whiteboard is an optimization parameter to be considered. An interesting result [21, 31] is that in a graph (G, λ) under the $\mathcal{W}$HITEBOARD model the mobile agents can solve the same set of problems as in the more traditional message-passing [68] distributed network with the same topology of (G, λ).

Another spatially restricted and not temporally restricted model is the $\mathcal{T}$OKEN model [49], it is also one of the oldest ones. The difference between $\mathcal{T}$OKEN and $\mathcal{W}$HITEBOARD is that

in $\mathcal{T}$OKEN the information given is implicit. The mobile agents have in their possession one or more tokens (also mentioned as *pebbles*) and when a mobile agent is at a node $v \in V$ of the network it can request access to the tokens at v respecting the fair mutual exclusion restriction. When the mobile agent gets access, it can count the number of tokens at v and using that as information the mobile agent can decide whether to put down a subset of its tokens, pick up a subset of the tokens on v or do nothing. There are also the *unmovable token* and the *enhanced token* (see [49]). The former is a stricter model and the latter is a more powerful model.

The $\mathcal{W}$IRELESS model is a model with temporal restrictions and no spatial restrictions. Essentially, the mobile agents can communicate with each other no matter where they are located in the network. The only restriction is that there is a specific point in time when the agents must be active in order to communicate. Usually this model assumes that the agents cannot "see" the other agents even if they are on the same node [49]. This model has been used e.g. in [36, 52].

The $\mathcal{B}$EEPING model [49] has the same restrictions, temporal but not spatial, as $\mathcal{W}$IRELESS, but the difference is that the information is implicit. It was first introduced in [25] and in this model again the agents are not visible to each other, even when they are on the same node. Each agent can transmit a signal called *beep*. If there are other agents active at that time, they hear that beep, but not if they are transmitting themselves, i.e. an agent can either transmit or listen.

2.2.2 Mobile agent capabilities

Mobile agents may have different range of visibility [49]. For *local visibility* the agents can only read the port labels of the node they are at, and they may or may not detect if there are more than one agent at that node. Depending on their memory they could know the exact number of agents at that node. Another model is for the agents to be able to see at some range (i.e., at q hops from the node an agent is at), this model is called q-hop visibility and studied in, e.g., [7, 11, 59]. The visibility range $q \geq 1$ is independent of the size of the graph [49]. The last model is for the agents to be able to see the whole network as well as the position of every agent in the network, which is called *global visibility* [49].

Except from visibility, mobile agents may have other knowledge too. Based on the description in [49], we remind the reader some of the knowledge that might be given to the agents before they start executing their protocol. Depending on their memory the agents might know n (i.e., the number of nodes in graph G) or m (i.e., the number of edges in graph G) or both. They might also know k (i.e., the number of agents operating in the network), also as an alternative the agents might be given just an upper-bound on those numbers (i.e., n, m, and k) and not the exact information. Other knowledge that the agents might have is information about the network topology (for example if they operate on a tree, grid, hypercube, etc.), if they don't know the topology the agents might have sense of direction or they might have a full *map* of the network. Finally, we say that the agents have *full knowledge* if they are provided with a full map of the network and the initial location for each agent in the network.

Another capability that we have to consider is the memory of the mobile agents, a notion which is used throughout the literature (e.g., [49, 63, 72]). In order for the agents to be able to compute their next actions, they need memory which may or may not be *persistent* [49]. We say the memory is persistent if the mobile agents retain the information saved in their memory even when they are not active. There is also a model called *oblivious* [49] where the agents lose all memory when they deactivate. The memory of the agents can either be finite, or bounded by some parameter. We could say that a lower bound for an agent's memory is at least the space needed to represent its states, but in some cases the total memory needed by an agent might be dependent on the space necessary to give some more powerful capabilities to the agent (e.g., global visibility). In the problems studied in this thesis when an agent has constant memory it means that it has constant number of states. In any case optimizing the memory usage of the mobile agents is an assessment measure for the protocols executed by the mobile agents (i.e., space complexity).

One more capability that we have to consider is whether the agents have distinct identities [63]. Usually the agents do not have such distinct identities initially, such agents are called *anonymous* agents. However, the mobile agents may be able to assign such distinct identities during the execution of the protocol.

2.2.3 Time

The notion of *time* is quite important for the agents. Each agent having a clock can operate based on the ticks that clock makes. Even if all the agents have such a clock, that does not ensure that all the agents will move and operate synchronously [49]. In fact, the time spent by each agent moving, observing, interacting with the environment, and computing, might be different from agent to agent and different in different stages of their operation. Also, the time between two consecutive active stages for each agent might differ too. Two very important models, that have been extensively studied are $\mathcal{A}$SYNC and $\mathcal{S}$SYNC [49].

In the *semi-synchronous* ($\mathcal{S}$SYNC also known as just *synchronous*) time is divided in time units; some of the agents then are active on any specific time unit (or round). These agents can compute, interact with the environment, move etc. and all the other agents are inactive for the entire time unit. An adversarial *activation scheduler* chooses which agent is active and which is inactive. A fairness to the activation scheduler is enforced, for every agent $a \in \mathcal{A}$ and time t, where $\mathcal{A}$ is the set of mobile agents operating in the network, there is a time $t' \geq t$ on which agent a is activated (i.e., each agent is activated infinitely often). At the extremes of $\mathcal{S}$SYNC we have the *fully synchronous* ($\mathcal{F}$SYNC) model. Under this model, in each time unit all agents are activated at the same time. Another extreme is the *sequential* ($\mathcal{S}$EQUENTIAL) model. Under this model at each time unit only one agent is activated.

On the contrary, in the *asynchronous* ($\mathcal{A}$SYNC) model there is no notion of time between the agents. There is no assumption as to how much time it takes for an agent to operate, but any agent can be active for only a finite amount of time. The agents are activated by the activation scheduler totally independently from one another, but the scheduler is said to be fair as mentioned before (i.e., each agent is activated infinitely often). Notice that even though the activation scheduler is fair in both the $\mathcal{A}$SYNC and the $\mathcal{S}$SYNC models, the number of activations of other agents between two consecutive activations of the same agent a might be unbounded. There is though a stronger assumption of a p-bounded (or p-fair) activation scheduler. Under that assumption, for every agent $a \in A$, between two successive activations of agent a, every other agent has been activated at most p times. Note that, in $\mathcal{F}$SYNC, the scheduler is 1-bounded by definition.

Time is also one of the assessment measures for any algorithm (i.e., time complexity).

2.3 Adversary

In the analysis of algorithms in order to prove correctness of an algorithm and show upper and lower bounds on time complexity as well as proving infeasibility under certain models, the abstract notion of an *adversary* is often used. In the case of mobile agents, the analysis of a problem can be thought of as a game between a proposed algorithm executed by mobile agents and the *adversary*. The adversary may have "powers" which it utilizes in order to prevent the mobile agents from executing the algorithm successfully. The powers of the adversary depend on how weak the model is. All parameters of an instance of a problem can be decided by the adversary. For example, the initial positions of the agents, the network topology, the time needed for a mobile agent to traverse an edge in the $\mathcal{A}$SYNC model, the number of the agents, the size of the network. We say that the problem is infeasible if the adversary can set the parameters in such a way to make any algorithm fail, and when the problem is feasible then the algorithm that solves the problem works for any options of the adversary.

2.4 Problems

In this section we discuss the main problems studied in the present thesis. Specifically, we have three main problems which are studied, namely the *gathering* problem, the *broadcast* problem, the *black virus decontamination* problem. We will formally define all details of the problems later in the relevant chapters.

2.4.1 Gathering

Consider a set of mobile entities that are able to move in an environment and whose task is to meet at the same location. This is a fundamental problem in the area of distributed computing with mobile agents (or robots), since agents, to execute specific tasks, may need to meet to share information or to coordinate. The problem is called *rendezvous* when the mobile agents are only two, *gathering* otherwise [65], and has been widely studied when the environment is modelled as a graph and the mobile agents can move

along the edges of the graph. However, most of the studies are restricted to fault-free environments and little is known about gathering when the environment is faulty or hostile. Possible faults can be a permanent failure of a node, like for example the so called *black hole* that destroys agents arriving at a node [6,40], or, transient faults that can appear anywhere in the graph and are controlled by a mobile hostile entity (an *intruder*) that behaves maliciously [8].

We investigate the gathering of multiple honest agents scattered in a graph in the presence of a malicious adversary as discussed in Section 2.3. We assume that the honest agents are much weaker than the malicious agent.

Problem 1. *Given a graph $G = (V, E)$ and k honest mobile agents scattered at distinct vertices on the network, find a protocol that gathers all agents at a single vertex.*

2.4.2 Black virus decontamination

We study the *Black Virus Decontamination* (BVD) problem within the distributed computing and especially the mobile agents' area. A Black Virus is a malicious entity and the goal is to find the minimum number of agents that can decontaminate a given network with a Black Virus initially located at an unknown place and design a fast distributed algorithm for a certain (preferably weak) model of mobile agents. In our study specifically we discuss the problem on rings but the problem can be defined in a general graph environment.

Problem 2. *Given a graph $G = (V, E)$ and k mobile agents scattered at distinct vertices of the graph, and a vertex of G unknown to the agents which is contaminated by a black virus, find a protocol that if followed by all mobile agents G gets decontaminated.*

2.4.3 Broadcasting

We are interested in communication problems for mobile agents moving in a network. The classical problems of broadcast or convergecast deal with the dissemination of information in the network. In the case of message passing networks, broadcast is achieved by spreading the information from the source node to all other nodes. For a system of mobile agents, the equivalent problem is the propagation of information from one source

agent to all other agents in the system. Such problems are relevant for teams of mobile sensor robots sent on data collection missions. We assume that the agents autonomously move along the edges of a graph that represents the network; when two agents are at the same node, they can communicate and share information. We would like to stress here that the agents are not allowed to use any means of communicating at a distance (e.g., due to security reasons). The information to be broadcast can be transferred only when the agents meet physically.

We define formally the problem of broadcasting bellow:

Problem 3 (The broadcast problem as defined in [37])**.** *Given a constantly connected dynamic network $\mathcal{G}$ based on an underlying graph G consisting of $n \geq 2$ nodes, a source agent that has a message $\mathcal{M}$ and $k \geq 1$ other agents that are initially located at distinct nodes of the network, the goal is to broadcast this message $\mathcal{M}$ to all the agents.*

3 Gathering of Mobile Agents

The problem of gathering two or more agents, in a graph is an important problem in the area of distributed computing and has been extensively studied especially for the fault free scenario. In this chapter we consider the mobile agents gathering problem in the presence of an adversarial *malicious agent* which by occupying an empty node prevents honest agents from entering that node. The honest agents move in synchronous rounds and at each round an agent can move to an adjacent node only if this node is not occupied by the malicious agent. We model the agents as finite state automata moving in an environment modeled by an anonymous oriented grid graph and having no information on the size of the graph. The malicious agent is assumed to be arbitrarily fast and to have full knowledge of the graph, the locations, and the strategy of the honest agents at all times. The honest mobile agents are anonymous, synchronous, cannot leave messages at nodes and they have constant memory. Two agents can see or communicate with each-other only when they meet at a node. Previous studies consider the problem for ring networks and for asynchronous grids, where rendezvous was solved only for the special case of agents starting already in connected configurations. We study here the problem for synchronous agents in anonymous oriented grid networks for any starting configuration without multiplicities and any number of agents. We first show that rendezvous is impossible for 2 agents even when the agents can see the locations of each-other at all times. We then present a universal deterministic algorithm that solves the problem for any number of at least 3 agents with only local visibility and constant memory in any oriented grid with one malicious agent. Some of those results have appeared in [34]

3.1 Mobile agents with global visibility

We first study the case where the agents have global visibility. In this scenario the agents are able to determine the locations of all other agents on the grid at all times. We first show some basic properties.

Let us consider a two dimensional oriented grid consisting of n rows and m columns. Port labels have been consistently assigned at each vertex:

$$\{north, south, east, west\}$$

and analogously for vertices on the border.

Algorithm 1: Solving the trivial case of no malicious agent.

```
1  while not on the north border do
2      move north
3  while not on the east border do
4      move east
```

Notice that if there is no malicious agent, then any number of agents with constant memory, having no initial information about the size of the grid, can easily gather within at most $n + m - 2$ steps by executing Algorithm 1. Eventually, each agent moves to the North-East corner of the grid. It is shown below that when there is a malicious mobile agent on the grid, a similar strategy can gather at least $(k - 1)$ out of the k agents. In this strategy when the agents are blocked by the malicious agent then they diverge a little and all except one can reach the meeting point.

Procedure MoveToNECorner

```
1  repeat
2      Compute S
3      Let A be the agent such that, x(A) = min_{∀A_i∈S} x(A_i)
4      S = S − A
5      if S = ∅ then Compute S without considering A
6      Let B be the agent such that, x(B) = min_{∀A_i∈S} x(A_i)
7      if A not at the north border then moves one step north
8      else moves one step east
9      if B not at the east border then moves one step east
10     else moves one step north
11 until there exists at most one agent A′ such that x(A′) + y(A′) > 1
```

Let us now discuss an algorithm that gathers $k-1$ agents with global visibility in the presence of a malicious agent. The idea of the algorithm is the following: The goal of that algorithm is to gather all but one agents at the north-east corner, or if the corner is occupied by the malicious at a neighboring node (i.e., the node immediately at the west or at the south of the corner). To achieve that, the agents would chose at each step two agents that are the closest to the corner but not at the corner or the adjacent nodes of the corner. The choice is made by selecting the agent which is closest to the north border as agent A and the second closest to the north border agent B. If there are ties, we break them by using the distance from the east border[1]. The agents that have been chosen make a move, one towards the north border and one towards the east border. At least one of them will reach the border and when they do, they start moving towards the north-east corner until they reach one of the adjacent nodes of the corner. When all but one have reached one of the two adjacent nodes of the corner, they can collectively decide at which node the should gather.

Algorithm 2: Gathering of $\geq k-1$ agents with global visibility.

```
1  Execute MoveToNECorner
2  if less than k − 1 agents are collocated then
3      if there are at least two agents A_1, A_2 with y(A_1) = y(A_2) = 0 then
4          Select A_1, A_2 (with smallest ids)
5          A_1 moves east and A_2 moves south
6          while ∃ agent C : y(C) = 0 do
7              moves through connected path south of (0, 0)
8          if (1, 1) is occupied then that agent moves east
9      else
10         if there are at least two agents A_1, A_2 with x(A_1) = x(A_2) = 0 then
11             Select A_1, A_2 (with smallest ids) and x(A_1) = x(A_2) = 0
12             A_1 moves north and A_2 moves west
13             while ∃ agent C : x(C) = 0 do
14                 moves through connected path west of (0, 0)
15             if (1, 1) is occupied then that agent moves north
16         else there is one agent at (0, 1) and one at (1, 0)
17             Both agents move towards (0, 0) and if blocked move towards (1, 1)
18     if (0, 0) is occupied then all agents move to (0, 0)
```

We define here some additional notation used in Algorithm 2. Let $(0,0)$ be the north east corner for description purposes, $y(A)$ be the distance of mobile agent A from the north

[1]Since the agents start at distinct nodes, one of those distances must be different.

border, $x(A)$ be the distance of mobile agent A from the east border, and the set of the agents who are closer to the north border, but not at nodes $(0,0)$, $(1,0)$, or $(0,1)$, is defined as

$$S = \{A_i \mid y(A_i) = \min_{\forall C} y(C), \text{ and } y(C) + x(C) > 1\}$$

where $(1,0)$ and $(0,1)$ are the neighboring nodes of $(0,0)$ to the west and south respectively.

Lemma 3.1. *In an oriented grid of size $n \times m$, with $n,\ m \geq 3$, $k \geq 3$ honest agents, initially placed at distinct nodes, and one malicious agent, at least $(k-1)$ honest agents executing Algorithm 2 can gather under the global visibility model.*

Proof. Notice that initially the agents can assign distinct identities to each agent, for example using the combination of their distances from the east and north borders. In view of Lemma 3.3 at least $k-2$ agents are located at nodes $(1,0)$ and $(0,1)$ collectively. If there are at least two agents at $(1,0)$, then two of that group is chosen (for example, two with the smallest identities) and they try to move south and east respectively. At least one agent succeeds effectively creating a "bridge" towards $(0,1)$. Then the rest of the agents move through the occupied by an agent neighboring node (i.e., the bridge) towards $(0,1)$ and finally the agent or agents, that moved initially, move to $(0,1)$. Finally, if $(0,0)$ is occupied then all agents move there and $k-1$ agents have gathered. If $(0,0)$ is not occupied then there were $k-1$ agents initially at nodes $(1,0)$ and $(0,1)$ collectively, and thus all $k-1$ agents have gathered at $(0,1)$.

If there is only one agent at $(1,0)$ then there are at least two agents at $(0,1)$ except from two special cases which we will consider later. The agents at $(0,1)$ form a bridge towards node $(1,0)$ and with the same argument as before at least $k-1$ agents gather either at $(1,0)$ or $(0,0)$.

The two special cases arise when $k=3$ or $k=4$. For $k=3$, there might be one agent at each node $(1,0)$ and $(0,1)$ respectively. If the malicious agent blocks the third agent then $k-1$ agents can gather at $(0,0)$ (the same happens if there was an agent at $(0,0)$ from

the beginning). For $k = 4$, in order to only have at most one agent in both $(1, 0)$ and $(0, 1)$ nodes then $(0, 0)$ must be occupied, which is the node where $k - 1$ agents gather. □

Lemma 3.2. *Let agents A and B be the agents chosen by Algorithm 2. At least one of agents A and B executing Algorithm 2 reaches node $(0, 1)$ or $(1, 0)$ after a finite amount of time.*

Proof. The agents A and B are chosen depending on their distances from the north and east borders. Since the initial placement of the agents is at distinct nodes, each pair of agents (A, B) would have either $y(A) \neq y(B)$ or $x(A) \neq x(B)$ or both. Moreover, agent B who is closer to the east border moves east and agent A moves north, their paths towards the nodes $(1, 0)$ and $(0, 1)$ for A and B respectively do not intersect, and because they are the closest agents to those nodes they do not meet any other agent. If there was an agent in any of those paths of A and B, then that agent would have been chosen as A or B.

The malicious agent can block only one of the agents A and B during that procedure. Notice that blocking one of them does not change their states, A remains A and B remains B. Therefore, after a finite amount of time if A (resp. B) is not blocked, then it reaches node $(1, 0)$ (resp. $(0, 1)$). □

Lemma 3.3. *After a finite amount of time at least $k - 2$ agents that execute Algorithm 2 are located at nodes $(0, 1)$ and $(1, 0)$ collectively.*

Proof. In view of Lemma 3.2 at least one agent reaches either $(1, 0)$ or $(0, 1)$. Without loss of generality, let A be the agent that reaches $(1, 0)$. If A is the only agent that managed to reach its node, then it has $x(A) + y(A) = 1$, thus on the next round a new pair of agents A and B is chosen. If there are two agents A and B that reach nodes $(1, 0)$ and $(0, 1)$ respectively, then both agents have $x(A) + y(A) = 1$ and $x(B) + y(B) = 1$, hence on the next round a new pair of agents A and B is chosen.

Applying at each phase Lemma 3.2 eventually at least $k - 2$ agents would have gathered at nodes $(1, 0)$ and $(0, 1)$ collectively. The two agents C_1, C_2 that might be missing is one that has $x(C_1)+y(C_1) > 1$ and is indefinitely blocked by the malicious agent and the other which has $x(C_2) + y(C_2) = 0$ □

Let us now recall the following property of a grid graph:

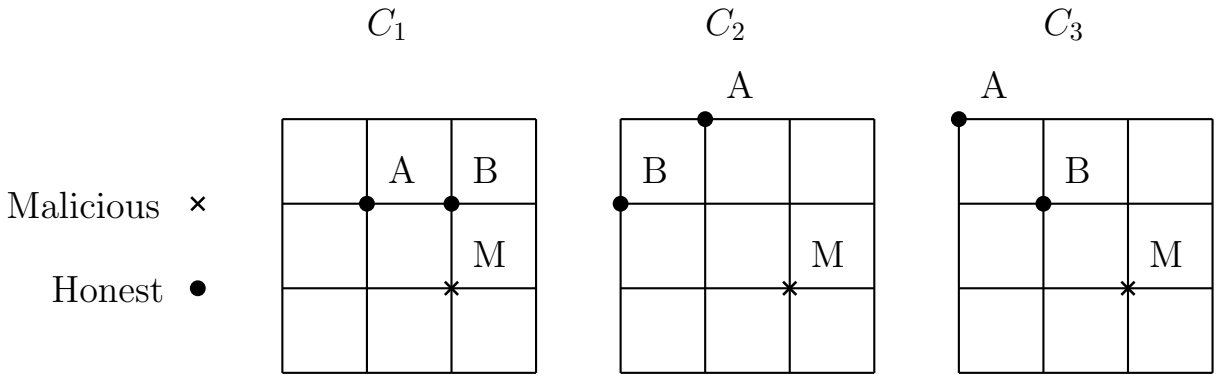

FIGURE 3.1: Two agents gathering configurations C_1, C_2, and C_3. The two agents are denoted by letters A and B and the malicious agent is denoted by the letter M.

Definition 3.1 (*vertex cut* [77, 78])**.** *A vertex cut, also called a vertex cut set or separating set, of a graph G is a subset of the vertex set $S \subseteq V(G)$ such that $G - S$ has more than one connected component.*

Definition 3.2 (*minimum vertex cut* [76, 78])**.** *Given a graph G a minimum vertex cut is a vertex cut of smallest possible size.*

Property 3.1. *An $n \times m$ grid graph $G = (V, E)$, with $n, m \geq 3$ has a minimum vertex cut of size 2, and every minimum vertex cut consists of the occupied nodes of either configuration C_2 of Fig. 3.1, or a symmetric one (all the pairs of nodes which are neighbours of a corner of the grid).*

3.1.1 Impossibility result for two honest agents

We first show that two agents cannot gather in an oriented grid, even if they have unlimited memory and global visibility. Notice that the agents can assign themselves distinct identities. More specifically, each agent can assign an identity to itself consisting of two integers by observing the distances from the borders. The first integer will be its distance from the north border, and the second integer will be its distance from the east border. Since the agents have global visibility and they have been initially placed at distinct nodes they can compute those distinct identities.

Lemma 3.4. *In an oriented grid of size $n \times m$, where $n, m > 3$, two agents cannot gather in the presence of one malicious agent, even if they have unlimited memory and global visibility.*

Proof. Let $G = (V, E)$ be an oriented grid. Suppose that there is an algorithm $\mathcal{A}$ that gathers the two agents at a node of the grid at time t. Consider the last configuration of the agents before they meet (i.e., the configuration at time $t-1$). We first prove that the last configuration should either be the one in which: 1) the two agents are adjacent somewhere in the grid (let us call this configuration C_1) or, 2) they are both at distance one from the same corner of the grid (let us call this configuration C_2). Examples of those two configurations are shown in Figure 3.1. Suppose for the sake of contradiction, that the last configuration C is not C_1 or C_2. In that case the agents in C have to be at distance two (but not as in C_2) at time $t-1$, otherwise they cannot meet at time t. This means that both agents have to move to the same (free) node u in order to meet. From Property 3.1, if $n, m \geq 3$, we have that the only vertex cut of size less than 3 is a vertex cut of size 2 which is the one of configuration C_2, or the symmetric ones. Thus, in all other configurations, the subgraph induced by all free nodes (i.e., nodes which are not occupied by honest agents) is connected. Hence, in that case the malicious agent M can always reach the free node u before the agents and thus prevent them from meeting each-other, even if the agents, have unlimited memory and they can see each other's locations. Thus, the last configuration before the gathering should be C_1 or C_2.

Let us now define configuration C_3 in which one of the agents occupies a corner node u of the grid and the other agent occupies the node v which is at distance 2 from u and not in the same row or column with u. We will show that unless the agents initially start in configuration C_1, C_2, or C_3, it is impossible to form any of those configurations, and hence it is impossible to gather. Suppose for the sake of contradiction that the agents are able to form a configuration of type $C \in \{C_1, C_2, C_3\}$ starting from a different configuration and let $C' \notin \{C_1, C_2, C_3\}$ be the last configuration before C is formed.

First observe that since C' is different than configuration C_2 and its symmetric ones, according to Property 3.1 configuration C' cannot have a vertex cut smaller than 3. Hence, if both agents at configuration C' try to move to the same node z, or only one agent tries to move to a node z in order to form configuration C, then M can reach node z before the agent(s) (since, in any configuration apart from C_2, M is able to reach any node which is not occupied). Therefore configuration C cannot be formed if at C' only one agent tries to move, or if both agents try to move at the same node. Let us study now the remaining

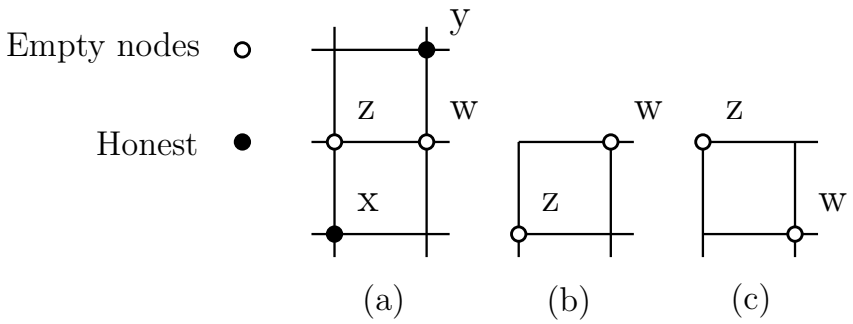

FIGURE 3.2: Two agents in a $n \times m, n, m > 3$ grid trying to move to two distinct nodes (z, w) from two distinct nodes (x, y) not in a configuration of type $\{C_1, C_2, C_3\}$ respectively. Nodes (z, w) are not occupied nodes, but need to be in order to get a configuration of type: $C_1,\ C_2,\ C_3$.

case in which both agents at C' try to move to two distinct nodes.

Consider a configuration $C' \notin \{C_1, C_2, C_3\}$ composed of a node x containing agent A and a node y containing agent B. Suppose that the two agents A, B located at nodes x, y in configuration C' are trying to move to two distinct nodes z and w respectively in order to form configuration C. If z, w are not the occupied nodes of a configuration $C \in \{C_1, C_2, C_3\}$, then the malicious agent M does not block anyone and therefore C cannot be formed. If z, w are indeed the occupied nodes of a configuration $C \in \{C_1, C_2, C_3\}$ but either the pair (x, w) or the pair (z, y) are not the occupied nodes of a configuration $C \in \{C_1, C_2, C_3\}$, then the malicious agent M could block either node z (so that the new occupied nodes are (x, w)) or node w (so that the new occupied nodes are (z, y)) respectively. Hence, again the malicious agent M has a strategy to prevent the agents from forming configuration C.

The only remaining hypothetical scenario in which the malicious agent M would not be able to prevent the formation of configuration C from C' is when all pairs of nodes (x, w), (z, y), and (z, w) are the occupied nodes of configurations in $\{C_1, C_2, C_3\}$. We show below that this is impossible.

- Suppose that the nodes (z, w) (in configuration C) are the occupied nodes of configuration type C_1. In other words, if both agents A, B move then the resulting configuration is connected (see an example in Figure 3.2(a)). Node x could not be at w, since then configuration C' would be of type C_1. Node x can only be either North or South of w in Figure 3.2(a), since otherwise (x, w) cannot be the occupied nodes of some configuration in $\{C_1, C_2, C_3\}$. Suppose without loss of generality that x is

as shown in Figure 3.2(a). Then y cannot be adjacent to x (otherwise configuration C' would be of type C_1) and can only be as shown in Figure 3.2(a), since otherwise (z, y) cannot be the occupied nodes of some configuration in $\{C_1, C_2, C_3\}$. However, even in that case, the pairs of nodes (x, w) and (z, y) cannot be the occupied nodes of configuration types C_2 or C_3 if $n, m > 3$.

- Suppose that the nodes (z, w) (in configuration C) are the occupied nodes of configuration C_2 (see an example in Figure 3.2(b)). Then nodes x, y have to be the other two nodes of the corner 2×2 subgrid in Figure 3.2(b), otherwise (x, w) or (z, y) cannot be the occupied nodes of some configuration in $\{C_1, C_2, C_3\}$. However, this means that configuration C' was of type C_3, which is a contradiction.

- Suppose that the nodes (z, w) (in configuration C) are the occupied nodes of configuration C_3 (see an example in Figure 3.2(c)). Then nodes x, y have to be the other two nodes of the corner 2×2 subgrid in Figure 3.2(c), otherwise (x, w) or (z, y) cannot be the occupied nodes of some configuration in $\{C_1, C_2, C_3\}$. However, this means that configuration C' was of type C_2, which is a contradiction.

Hence, if the two agents initially start at a configuration of type different than C_1, C_2, or C_3, then they cannot form a configuration of type C_1, C_2, or C_3, and therefore they cannot gather. Notice that, this impossibility result holds even when the agents have unlimited memory and can see each-other's location at any time on the grid. □

We will see later that 3 or more agents can always gather in this model.

3.1.2 Gathering any number of at least 3 agents

In this section, we show that under the global visibility model, even three honest agents (with constant memory) can gather in an oriented grid in presence of a malicious agent.

Some notations:

- Let C_0^3 be the set of all connected configurations with 3 agents (i.e., the nodes occupied by the agents form a connected subgraph of the grid).

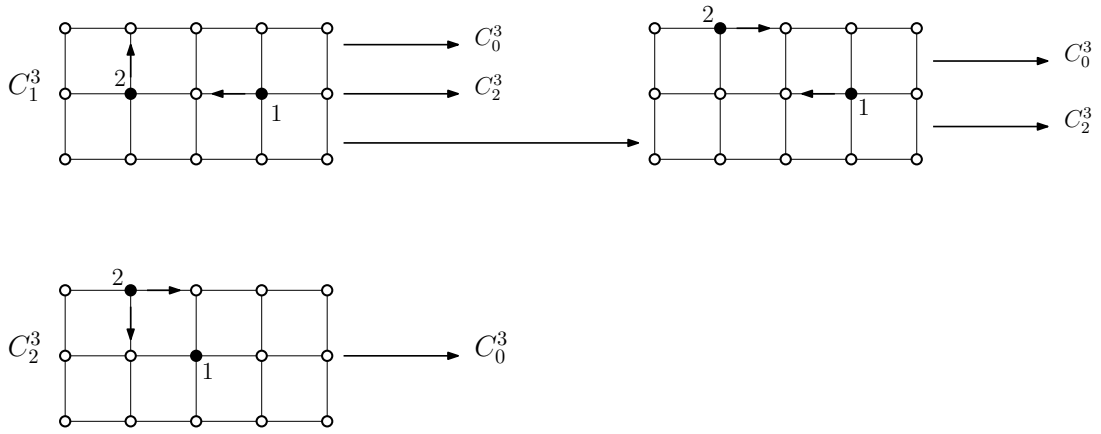

FIGURE 3.3: Three agents in a grid: A tower of 2 and a single agent at distance two.

- Let C_1^3 be the set of all configurations with 3 agents, where two agents are colocated and the third agent is at distance two from them on a straight line (i.e., either on the same row or on the same column, see an example in Fig. 3.3).
- Let C_2^3 be the set of all configurations with 3 agents, where two agents are colocated and the third agent is at distance two from them not on a straight line (i.e., the agents are not all on the same row or on the same column, see an example in Fig. 3.3).

Theorem 3.1. *Three honest agents with global visibility can gather in an oriented grid in spite of one malicious agent.*

Proof. It is sufficient to show that the agents can form a connected configuration (it is straightforward to gather from a connected configuration if the agents can see each other). Due to Lemma 3.1, we know that 2 of the 3 agents can always gather at a node, if they have global visibility. So, let us assume that we start from a configuration where 2 agents are colocated (form a tower) and the third agent is in some distinct node of the grid. Due to the global visibility capability, the agents can approach each other, i.e., they can try to move to reduce the (vertical and then horizontal) distance between them. The two agents in the tower will move together during this process. Note that, if the distance between them is more than two then M can block either the tower or the third agent but not both at the same time. Thus, at each time step, the distance will be reduced until the distance is no more than two. If the distance is less than two then the agents already form a connected configuration and they can immediately gather. So, suppose that the

agents reach a configuration where the distance between the tower and the third agent is exactly two.

Thus this configuration can be either of type C_1^3 or of type C_2^3. We show that: (i) From a configuration of type C_1^3 we can reach a configuration of type C_2^3 or a connected configuration, and (ii) from a configuration of type C_2^3 we can always reach a connected configuration.

To prove (i), let us consider, w.l.o.g., the particular configuration $C \in C_1^3$ where the third agent is two steps to the EAST of the tower of two agents (see Fig. 3.3). Note that for the other configurations in C_1^3, similar arguments hold, with rotation of directions etc. In configuration C, the algorithm will instruct the tower to perform GO(NORTH) and the solitary agent to perform GO(WEST). If both moves succeed, then the resulting configuration is in C_2^3 and we are done. If only the move of the tower is blocked then we have a connected configuration. So, we need to consider the only other case where the move of the solitary agent is blocked by M. The resulting configuration has the tower one step North and two steps West of the solitary agent (as shown in the figure). From this configuration, in the next step, the algorithm will instruct the tower to perform GO(EAST) and the solitary agent to perform GO(WEST). If both moves succeed then we have a connected configuration and if either one of the moves is blocked then the resulting configuration is in C_2^3. Thus we have proved (i).

To prove (ii), note that in any configuration of type C_2^3, there are two unoccupied nodes of the grid that are both at distance one from the tower and from the solitary agent. The algorithm will instruct the agents in the tower to split and move towards those two nodes respectively[2]. The malicious agent M cannot block the moves of both and if at least one of the moves succeeds then the resulting configuration is a connected configuration. Thus we have proved (ii). So, three agents with global visibility can always gather starting from any configuration. □

Notice that the above result can be extended to any number of k agents, by first forming a tower of $(k-1)$ agents (cf., Lemma 3.1) and repeating the technique of Theorem 3.1 with the $k-2$ agents of the tower acting as a single agent. However, we will show in

[2]We remind the reader, that as we have noticed in the beginning of the section, the agents can assign to themselves distinct identities and therefore they can perform distinct moves.

the next section that for $k > 3$ agents, gathering is still possible in the more challenging model with local visibility.

3.2 Local visibility

3.2.1 Gathering any number of at least 4 agents

In this section, we show that four or more honest agents can always gather in an $n \times m$ oriented grid with one malicious agent, where $n, m > 1$. Notice that, if n or m equals 1 (i.e., the grid is just one row or column), the gathering of all honest agents is trivially impossible since the malicious agent can divide the agents into two separate groups and prevent them from gathering. Hence we are interested in $n \times m$ oriented grids, where $n, m > 1$. Notice also that, if in the starting configuration there are nodes with more than one agents, then those initially colocated agents will always behave in the same way since they are anonymous deterministic automata acting synchronously and they always have the same input. Thus, for an initial configuration with two colocated groups of agents the problem is unsolvable since the two groups will act as two agents and Lemma 3.4 will apply. Hence, in general the problem is unsolvable for initial configurations with multiple agents at distinct nodes. Therefore, we are interested in initial configurations with scattered agents (i.e., at most one agent at each node).

The high level description of the algorithm is the following: The agents first form at least one group of at least 3 colocated agents. Each such group moves to the South-West corner of the grid. Then, each group of at least 3 agents sweeps the whole grid towards the North-East corner and on the way collects all the agents not belonging to a group. Finally, all agents meet either at the North-East corner of the grid or at its adjacent node on the West.

The main algorithm, called `GridWalk`, first creates at least one group of three colocated agents, called tower, and then moves the towers of agents towards the South-West corner of the grid. After arriving at the South-West corner the towers sweep the grid towards the North-East corner, by exploring each column and changing columns only via the South border while on the way collect all the agents that do not belong to any tower. Note that, when two (or more) agents are moving at the same time to different nodes, the malicious

agent M can block at most one of them at each step, since the movements of the agents are synchronous. An agent can move in a chosen direction only if the neighbouring node is not occupied by M, otherwise it stays (we say it is blocked) for this step.

Procedure FormTower

```
/* Algorithm to form Towers of at least 3 agents for k ≥ 4          */
/* Returns the state of the agent                                   */
State := Initial
while there are less than 3 agents at the current node and there is an edge North do
    if there is exactly one agent at the current node then
        if previous move was North and blocked and there is an edge East then move
          one step East
        else move one step North
    else
        // there are exactly two agents at the current node
        if previous move was North and blocked for both agents at state Initial then
            A1 moves one step North
            if there is an edge East then A2 moves one step East
            else A2 moves one step West
        else move one step North
State := NB-reached
while there are less than 3 agents at the current node and there is an edge East do
    if there is exactly one agent at the current node then move one step East
    else
        // there are exactly two agents at the current node
        if previous move was East and blocked for both agents at state NB-reached
          then
            A1 moves one step East
            if there is an edge North then A2 moves one step North
            else A2 moves one step South
        else move one step East
while there are less than 3 agents at the current node and there is an edge North do
    move one step North
while there are less than 3 agents at the current node do WAIT(1)
State := Tower
return State
```

The formal description of `GridWalk` is shown in Algorithm 3. The algorithm uses a number of procedures. The first one, is called Procedure FormTower and creates at least one tower of at least 3 collocated agents. The second one, is called Procedure TowerWalk and moves a tower of 3 agents to the next node in the intended direction. Finally Procedure

ExploreLine, is used to move a tower in a straight direction.

Let us now describe all procedures in more detail. The main idea of Procedure FormTower is to let each agent navigate towards the North-East corner, by first instructing the agent to move North up to the North border and then East up to the East border. During this navigation, if an agent is blocked by the malicious agent while moving North and it is alone, in the next move it will try to move East for one step (or West if it is located on the East border column). If at some point the agent (which still moves towards North) meets with exactly one more agent and both of them were blocked while trying to move North, then one of them moves East for one step (or West if it is located on the East border), while the other one moves North. If two agents which have already reached the North border are blocked on the North border while they try to move East, then one of the agents moves South for one step. The agents that reach the North-East corner wait until at least 3 agents are there.

Lemma 3.5. *Consider $k \geq 4$ agents in a $n \times m$ oriented grid, initially placed at distinct nodes, with one malicious agent, where $n, m > 1$. If the agents execute Procedure* `FormTower` *then at least 3 agents will meet at a node at the same time within $O(n+m)$ time units.*

Proof. Initially every agent is moving towards the North border of the grid. If an agent which is alone is blocked by the malicious agent M while trying to go North, and the agent is not on the East-border column, then it tries to go East for just one move, i.e., before continuing to try to go North. If there are two co-located agents which have been blocked while they are trying to go North, then one of them tries to go East (or West if they are moving on the East-border column) for just one move, while the other one tries to go North. Note that even though the agents are anonymous, since they initially started at distinct nodes, they can assign to themselves different labels when they gather at a node, based on the arrival times and on the incoming directions. Since M can only block at most one node at a time, it might prevent at most 2 agents from approaching the North border while executing the first `while-loop` of Procedure `FormTower`: M might prevent one agent A (or a group of at most two co-located agents) from going North and at the next time unit (when A tries to go East) it might prevent another agent B (or a group of at most two co-located agents) from going North; M can continue to prevent

agent A (when B goes East) and B (when A goes East) from going North. Hence at least $k-2$ agents will finish executing their first `while-loop` in the procedure approaching the North border of the grid. At least one of the remaining two agents will first reach the East border and then it will try to move North. Even if both remaining agents at some point occupy the same node on the East border, they will split, one going West, and the other North, hence at least one of them will manage to approach the North border.

Thus at least $k-1$ agents will exit the execution of their first `while-loop` in the procedure and reach the North border after $O(n+m)$ time units. Now every agent on the North border will execute its second `while-loop` trying to go East. The malicious agent M can divide those $k-1$ agents on the North border, into at most two groups and can prevent the western group from approaching the East border. Notice that, since $k-1 \geq 3$ at least one of these groups should have at least two agents.

If there is only one agent on the western group of the North border, then either it will be eventually unblocked, or otherwise the agent which was still blocked without exiting its first `while-loop`, will be unblocked and reach the North border, joining one of the two groups there.

If there are two agents on the western group of the North border, then those agents (by executing their second `while-loop`) will split after they are blocked at the same node (one of them continuing moving East and the other one moving South for one step). Hence, the malicious agent can only prevent at most one of them from moving East within the next two steps. Overall, at most one agent might not exit its first `while-loop` and at most one more agent might not exit its second `while-loop`. This case can only happen if the malicious agent has blocked the North-East corner of the grid. Otherwise, at most one agent might not exit its first `while-loop`, or at most one agent might not exit its second `while-loop`. Notice that the agents which have exited their first `while-loop` they are later located either in the North border or one row below. The agents which have also exited their second `while-loop` they are later located either in the North-East corner or one node below. Hence if $k-1 \geq 3$ agents have exited their second `while-loop` they meet and finish the procedure. If the malicious agent blocks the North-East corner then it might happen that one agent has not exited its first `while-loop` and is located one node South of the North-East corner and another agent has not exited its second `while-loop` and is

located one node West of the North-East corner. However, in such a case at least two of the remaining $k - 2 \geq 2$ agents will soon join one of the blocked agents. Notice that, in the special case where two groups of two agents each are temporarily formed, one West and one South of the North-East corner, two agents might swap positions once, but then at least three agents will gather at the node South of the North-East corner.

Hence at least one group of at least 3 co-located agents will be formed. □

Procedure TowerWalk(X, Y)

/* A tower of at least 3 agents moves from a node $u = (x, y)$ either to an adjacent node in direction X, or to a node at a distance 2 from u in direction XY. If at some step there are no applicable instructions for an agent, this agent does not move in that step. All other agents associated with the tower except the 3 basic agents, are just following one of the three basic agents. */

Step 1
- A_1 : move to $(x + 1, y)$
- A_2 : move to $(x, y + 1)$

Step 2
- A_3 : [together with A_1]: move to $(x, y + 1)$
- A_3 : [together with A_2]: move to $(x + 1, y)$

Step 3
- $A1$: [alone at (x, y)]: move to $(x + 1, y)$
- $A2$: [alone at $(x, y + 1)$]: move back to (x, y)
- $A2$: [alone at (x, y)]: move to $(x + 1, y)$
- $A2$: [otherwise]: move to $(x + 1, y + 1)$

Step 4
- $A1$: [alone at (x, y)]: move to $(x, y + 1)$
- $A2$: [together with $A3$ at $(x, y + 1)$]: move to $(x + 1, y + 1)$
- $A2$: [together with $A3$ at (x, y)]: move to $(x + 1, y)$
- $A3$: [together with $A2$ at $(x, y + 1)$]: move to (x, y)
- $A3$: [together with $A2$ at (x, y)]: move to $(x + 1, y)$

Step 5
- $A1$: [together with $A3$ at $(x, y+1)$]: move to $(x+1, y+1)$
- $A2$: [alone at $(x, y+1)$]: move back to (x, y)
- $A3$: [alone at $(x, y+1)$]: move to $(x+1, y+1)$
- $A3$: [together with $A1$ at $(x, y+1)$]: move to $(x+1, y+1)$

Step 6
- $A2$: [alone at $(x+1, y+1)$]: move to $(x+1, y)$
- $A2$: [together with $A3$ at (x, y)]: move to $(x+1, y)$
- $A2$: [together with $A3$ at $(x+1, y+1)$]: move to $(x+1, y)$
- $A3$: [alone at (x, y)]: move to $(x+1, y)$
- $A3$: [together with $A2$ at (x, y)]: move to $(x+1, y)$
- $A3$: [together with $A2$ at $(x+1, y+1)$]: move to $(x+1, y)$

Consider a number of at least three agents which gather at a node constructing a tower [3]. Even in the case that more than three agents gather at the same node, three agents can always be elected as the three distinct agents that form the tower (e.g., by comparing arrival times, incoming directions, already assigned identities and making an ordered list in which all agents agree). The remaining agents (if any) are then identified as followers of the constructed tower. Hence assume a tower of at least three agents which has been formed by agents $A1$, $A2$ and $A3$.

Procedure TowerWalk, moves all agents associated with the tower either to an adjacent node towards a given direction or to a node at distance 2 from the agents' current position. Let X be the given direction and Y (which is also given) be the direction 90 degrees clockwise or counterclockwise from X. For example if X is North then Y will be either East or West and so on. Suppose the tower is at position (x, y), where x is the current row-position and y is the current column-position. Let $(x+1, y)$ be the node which is adjacent to node (x, y) towards direction X and let $(x, y+1)$ be the node which is adjacent to node (x, y) towards direction Y. Then, after an execution of `TowerWalk(`X, Y`)`, the new position of the tower is either $(x+1, y)$ (if the node $(x+1, y)$ was not blocked) or $(x+1, y+1)$ (if the node $(x+1, y)$ was blocked). For example if at least 3 agents belong to a tower at a node u then Procedure `TowerWalk(North, East)` will move all agents associated with the tower either to node v adjacent to u in the North direction (if v is not blocked) or to node w at distance 2 from u in direction North-East (if v was blocked).

[3] We remind the reader that, even though the agents are anonymous, since they initially started at distinct nodes, they can assign to themselves different labels when they gather at a node, based on several criteria.

The formal description of Procedure `TowerWalk`(X, Y) can be found in Algorithm Tower-Walk. The algorithm has also been demonstrated in Figures 3.4, 3.5, 3.6 for $X = North$ and $Y = East$, the remaining cases for X and Y work in a similar way. In the algorithm, the related figures and correctness lemma we only analyze the movements of the three agents $A1$, $A2$ and $A3$ which formed the tower. If there are more agents associated with the tower then those agents just follow one of the three previously mentioned agents.

Lemma 3.6. *Consider $k \geq 4$ agents in a $n \times m$ oriented grid, initially placed at distinct nodes, with one malicious agent, where $n, m > 1$. Suppose that in the grid there is at least one tower consisted of at least $l \geq 3$ colocated agents located at a node $u = (x, y)$ at time t. If the tower-agents execute Procedure* `TowerWalk`(X, Y)*, then after at most 6 time units, they all move either 1) to a node $v = (x + 1, y)$ which is adjacent to u towards direction X, if v was not blocked at time $t + 1$ or $t + 3$ or, 2) to a node $w = (x + 1, y + 1)$ which is at a distance 2 from u towards direction $X - Y$, if v was blocked at time $t + 1$ and $t + 3$.*

Proof. We will prove the lemma for directions X, Y, where X is any direction and Y is the direction 90 degrees clockwise from X. The case when Y is the direction 90 degrees counterclockwise from X can be proved in a similar way. We have depicted the algorithm in Figures 3.4, 3.5 and 3.6, for $X = North$ and $Y = East$. We describe the algorithm by just illustrating and analyzing the movement of the 3 basic agents $A1$, $A2$ and $A3$ that formed the tower. If there are more agents associated with the tower then those agents just move together with one of the three previously mentioned agents. If one of the three basic agents meets at any time another agent C which has not yet been associated with any tower, then agent C is associated with the tower and after that it just moves together with one of the three basic agents of the tower. If one of the agents associated with the tower meets at any time an agent C which is associated with a different tower, then the agents just ignore each-other. Note that two tower-agents that meet at the same node can always decide whether they are associated with the same tower or not: Suppose that two agents A, B associated with towers started the execution of Procedure `TowerWalk` at nodes u_A, u_B respectively and meet at a node v while executing the procedure. The two tower agents communicate and compare their port-sequences from the starting nodes u_A, u_B to the meeting node v. Since, as we will prove, Procedure `TowerWalk` lasts at most 6 steps,

a tower-agent needs only constant memory to keep the sequence of ports from the node that started the execution of the procedure.

In the first step of the Procedure `TowerWalk` (X, Y), agents $A1$ and $A2$ move synchronously to different nodes and therefore the malicious agent can only block at most one of the agents. Agent $A3$ does not move. Thus the following three cases arise:

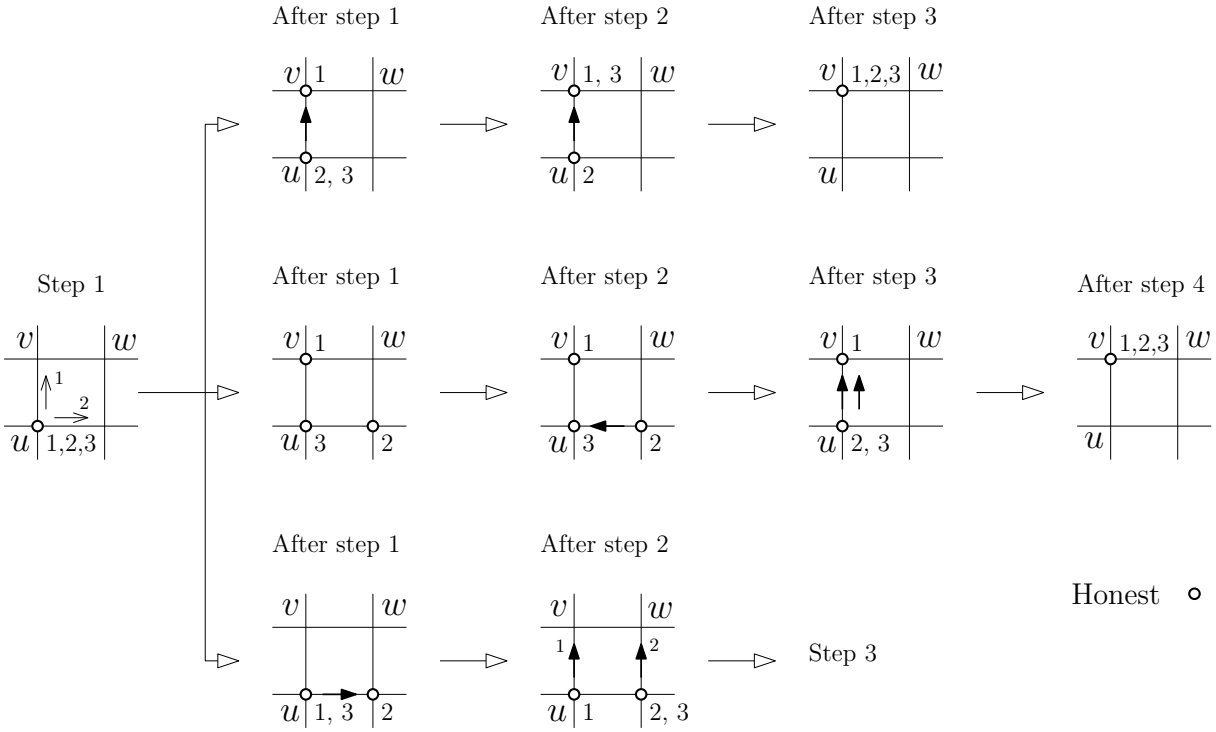

FIGURE 3.4: Consider three colocated agents with identities $1, 2, 3$ which have formed a tower and are located at node u. The agents execute Procedure `Towerwalk(North, East)`. The arrows depict the intended moves at step 1 on the left of the figure. Each possible configuration from step t to step $t + 1$ is shown under the title 'After step t'. In each configuration the arrows depict the agents' intended moves of the next step. For the series of the lower configurations, the demonstration continues in Figure 3.5.

- $A2$ is blocked at step 1, hence $A1$ moves to node $(x + 1, y)$ and $A3$ does not move. After step 1 $A1$ is at $(x + 1, y)$ and $A2$ is at (x, y) with $A3$, as shown in the first row of Figure 3.4. Therefore at step 2 $A1$ and $A2$ do not move and $A3$ moves to $(x + 1, y)$ where $A1$ is located. After step 2 $A1$ is with $A3$ at $(x + 1, y)$ and $A2$ is alone at (x, y) (first row, second column, of Figure 3.4). Therefore at step 3 $A1$ and $A3$ do not move, $A2$ moves to $(x + 1, y)$ where $A1$ and $A3$ are located. Therefore after step 3 $A1$, $A2$ and $A3$ are all together at $(x + 1, y)$ (first row, third column, of Figure 3.4).
- Both $A1$ and $A2$ move at step 1. Hence, after step 1 $A1$ is at $(x + 1, y)$ and $A2$ is at

$(x, y+1)$ and A3 is at (x, y), as shown in Figure 3.4, second row. Then at step 2 no one will move, since $A3$ is alone at (x, y), therefore after step 2 there is no changes at the locations (second row of Figure 3.4). At step 3 $A1$ and $A3$ do not move and $A2$ moves back to (x, y) to meet $A3$. After step 3 $A1$ is alone at $(x + 1, y)$ and $A2$ and $A3$ are at (x, y). Therefore at step 4, $A1$ does not move and $A2$ and $A3$ move to $(x+1, y)$ where $A1$ is located. Therefore after step 4 $A1$, $A2$ and $A3$ are all together at $(x + 1, y)$.

- $A1$ is blocked at step 1, hence $A2$ moves to $(x, y + 1)$. Therefore, after step 1 $A2$ is at $(x, y + 1)$ and $A1$ and $A3$ are at (x, y), as shown third row of Figure 3.4. In step 2 $A1$ and $A2$ do not move, $A3$ moves to $(x, y + 1)$. Hence after step 2 $A1$ is at (x, y) and $A2$ and $A3$ are at $(x, y + 1)$ (third row of Figure 3.4). At step 3 $A3$ does not move, $A1$ tries to move to $(x + 1, y)$ and $A2$ tries to move to $(x + 1, y + 1)$. Depending on which agent succeed in his/her movement, three sub-cases arise, which are presented in Figure 3.5:

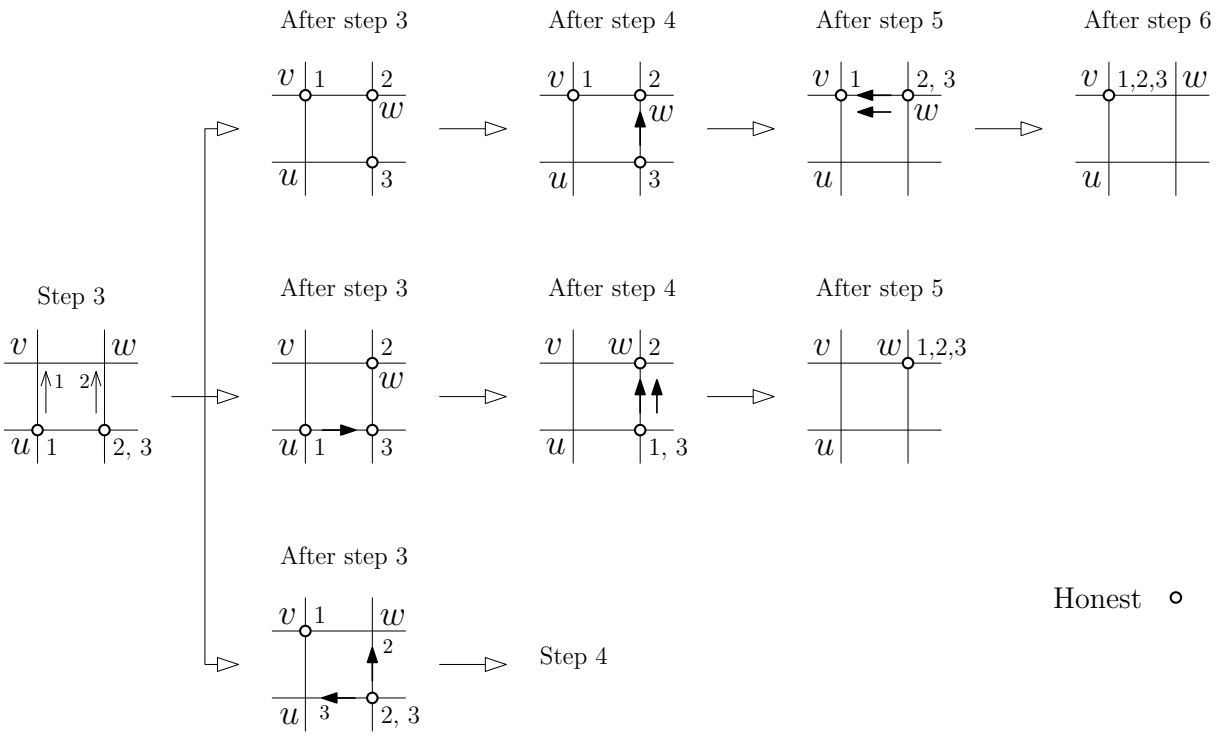

FIGURE 3.5: Continuation of the series of the lower configurations of Figure 3.4. For the series of the lower configuration, the demonstration continues in Figure 3.6.

  - Both $A1$ and $A2$ move. Hence after step 3 $A1$ is at $(x+1, y)$, $A2$ is at $(x+1, y+1)$ and $A3$ is at $(x, y+1)$, as shown in first row, Figure 3.5. At step 4 no one moves, hence after step 4 there is no change at the locations (first row, Figure 3.5). At

step 5 $A1$ and $A2$ do not move and $A3$ moves to $(x+1, y+1)$, where $A2$ is located. Therefore after step 5 $A1$ is at $(x+1, y)$ and $A2$ and $A3$ are at $(x+1, y+1)$ (first row, Figure 3.5). At step 6 $A2$ and $A3$ move to $(x+1, y)$, where $A1$ is located. Therefore after step 6 $A1$, $A2$ and $A3$ are all together at $(x+1, y)$.

– $A1$ is blocked, hence $A2$ moves to $(x+1, y+1)$. After step 3 $A1$ is alone at (x, y), $A2$ is at $(x+1, y+1)$ and $A3$ is at $(x, y+1)$, as shown in second row, Figure 3.5. At step 4 $A1$ moves to $(x, y+1)$, $A2$ and $A3$ are alone so they do not move. Therefore after step 4 $A1$ and $A3$ are at $(x, y+1)$ and $A2$ is at $(x+1, y+1)$ (second row, Figure 3.5). At step 5 $A2$ does not move, $A1$ and $A3$ move to $(x+1, y+1)$, hence after step 5 $A1$, $A2$ and $A3$ are all together at $(x+1, y+1)$.

– $A2$ is blocked, hence $A1$ moves to $(x+1, y)$. After step 3 $A1$ is at $(x+1, y)$ and $A2$ and $A3$ are at $(x, y+1)$, as shown in third row, Figure 3.5. At step 4 $A1$ does not move, $A2$ tries to move to $(x+1, y+1)$ and $A3$ tries to move back to (x, y). Again depending on which agent succeed in his/her move, three sub-cases arise which are presented in Figure 3.6:

  * $A3$ is blocked, hence $A2$ moves to $(x+1, y+1)$. After step 4 $A1$ is at $(x+1, y)$, $A2$ is at $(x+1, y+1)$ and $A3$ is at $(x, y+1)$, as shown in the first row of Figure 3.6. Therefore at step 5 $A1$ and $A2$ do not move and $A3$ moves to $(x+1, y+1)$, where $A2$ is located. After step 5 $A1$ is at $(x+1, y)$ and $A2$ and $A3$ are at $(x+1, y+1)$ (the first row of Figure 3.6). Hence at step 6 $A1$ does not move and $A2$ and $A3$ move to $(x+1, y)$, where $A1$ is located. Therefore after step 6 $A1$, $A2$ and $A3$ are all together at $(x+1, y)$.

  * $A2$ is blocked, hence $A3$ moves to (x, y). After step 4 $A1$ is at $(x+1, y)$, $A2$ is at $(x, y+1)$ and $A3$ is at (x, y), as shown in the second row of Figure 3.6. Therefore at step 5 $A1$ and $A3$ do not move and $A2$ moves back to (x, y), where $A3$ is located. After step 5 $A1$ is at $(x+1, y)$ and $A2$ and $A3$ are at (x, y) (second row of Figure 3.6). At step 6 $A1$ does not move and $A2$ and $A3$ move to $(x+1, y)$, where $A1$ is located. Therefore after step 6 $A1$, $A2$ and $A3$ are all together at $(x+1, y)$.

  * Both $A2$ and $A3$ move. Hence after step 4 $A1$ is at $(x+1, y)$, $A2$ is at $(x+1, y+1)$

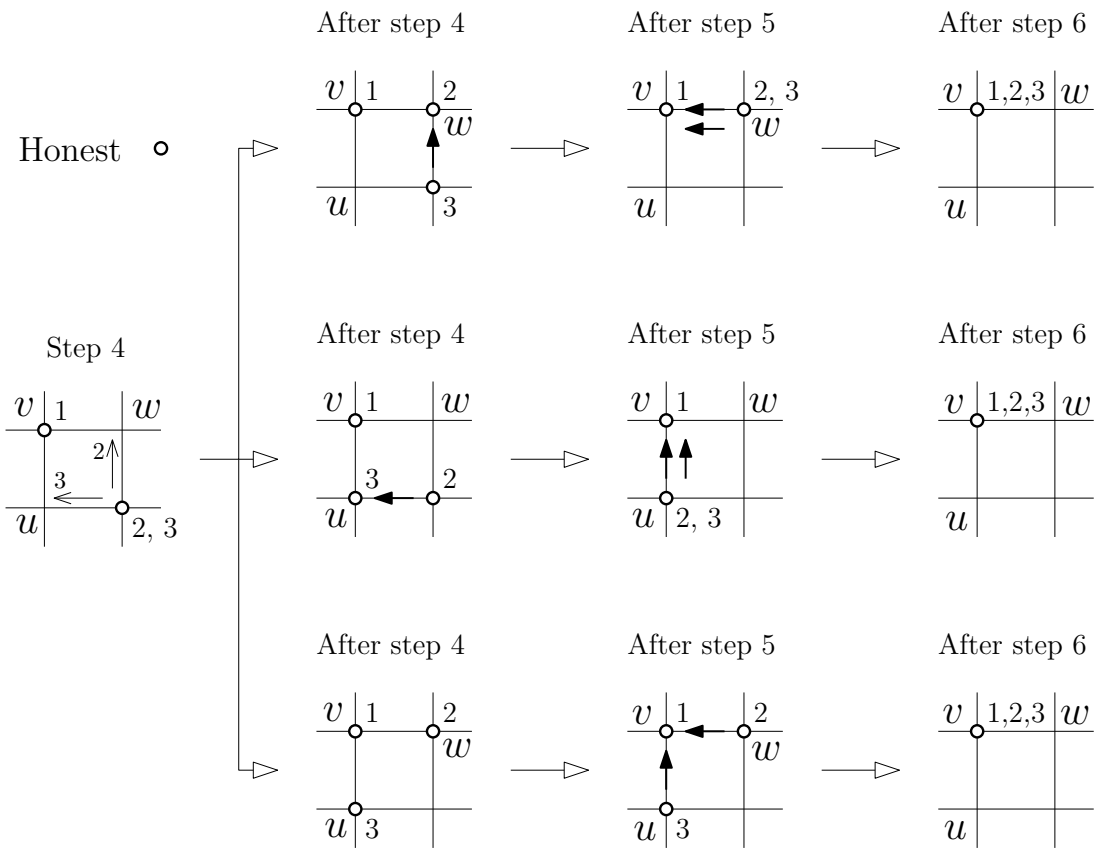

FIGURE 3.6: Continuation of the series of the lower configuration of Figure 3.5.

and $A3$ is at (x, y), as shown in the third row of Figure 3.6. At step 5 no one moves, hence after step 5 there is no change at the locations (third row of Figure 3.6). At step 6 $A1$ does not move and $A2$ and $A3$ move to $(x+1, y)$, where $A1$ is located. Therefore after step 6 $A1$, $A2$ and $A3$ are all together at $(x+1, y)$.

□

Procedure ExploreLine(Dir)

```
/* Algorithm for Exploring a line (row or column) of the Grid towards
   direction Dir                                                    */
```

1 $X = Dir$
2 Y is the direction 90 degrees clockwise from direction X
3 $parity = 0$
4 **while** *there is an edge towards direction* X **do**
5 **if** *there is no edge towards direction* Y **then**
6 Y is the direction 90 degrees counter-clockwise from direction X
7 Let u be the location of the tower
8 Perform TowerWalk(X, Y)
9 Let v be the location of the tower after executing Procedure TowerWalk
10 **if** *node* v *is at a distance* 2 *from* u *on the* XY *direction* **then**
11 **if** *there is an edge towards* X **then**
12 Let Z be the opposite direction of Y
13 Perform TowerWalk(Z, X)
14 **else**
15 $parity = 1$
16 **return** $parity$

Procedure `ExploreLine(`*Dir*`)`, moves a tower consisting of at least 3 agents from a node u either to a node v on the border of the grid in a straight direction Dir from u, or to a node v' on the border adjacent to v (if v was blocked at that time). The procedure repeatedly calls Procedure `TowerWalk(`X, Y`)`, where $X = Dir$ and Y is a direction 90 degrees clockwise or counterclockwise from X. If at some point the tower ends up at a node $w \neq v'$ which

is not on the straight direction connecting nodes u, v, then a suitable correction move is executed. For example, if the tower had to move North but ended up North-East (before it reaches the target border) then it moves West, thus going back to its track. This ensures that the tower always moves in a straight direction. Note that for parameter Dir we use only the directions of north, south, or west.[4]

Possible executions of Procedure `ExploreLine` are shown in Figures 3.7, 3.8.

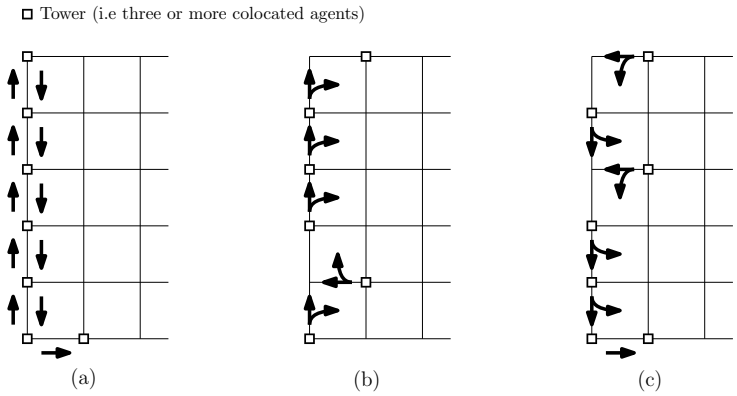

FIGURE 3.7: Possible executions of Procedure `ExploreLine`. (a) A tower traversing back and forth a column without been blocked (Procedure `ExploreLine(North)`, followed by Procedure `ExploreLine(South)` and Procedure `TowerWalk(East, North)`). (b) A tower traversing the West border column towards North (Procedure `ExploreLine(North)`). The tower was blocked two times and those times temporarily ended up in the next column. (c) A tower traversing the West border column towards South (Procedure `ExploreLine(South)`, followed by Procedure `TowerWalk(East, North)`). The tower was blocked once and that time temporarily ended up in the next column.

Lemma 3.7. *Consider $k \geq 4$ agents in a $n \times m$ oriented grid, initially placed at distinct nodes, with one malicious agent, where $n, m > 1$. Suppose that in the grid there is at least one tower consisted of at least $l \geq 3$ colocated agents located at a node u at time t. Let v be the node on the border of the grid towards direction Dir furthest from u and let p be the distance (i.e., the length of the shortest path) between nodes u and v. If the tower-agents execute Procedure ExploreLine(Dir), then after $O(p)$ time units the agents will end up either at node v (in this case the procedure returns $Parity = 0$) or at a node on the border adjacent to v (in that case the procedure returns $Parity = 1$) depending on the location of node u and direction Dir as follows:*

i) Dir is North and u is not on the East border: v or East of v

[4]Even though Dir could also get east as value, we do not use it in our algorithm since the move from one column to the next towards the east is made by only one execution of `TowerWalk`.

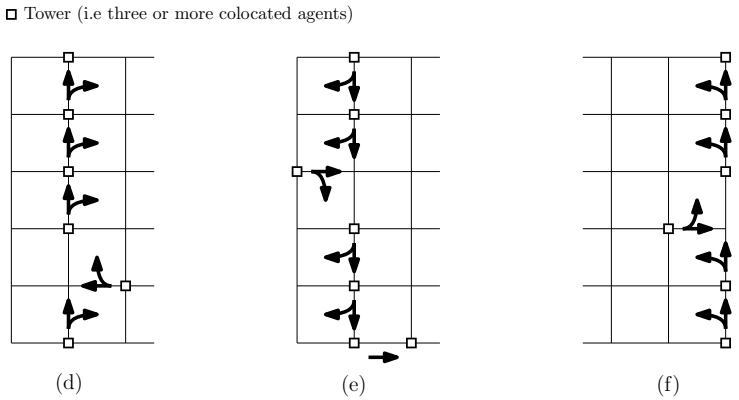

FIGURE 3.8: Possible executions of Procedure `ExploreLine`. (d) A tower traversing a non-border column towards north (Procedure `ExploreLine(North)`). The tower was blocked once and that time temporarily ended up in the next column. (e) A tower traversing a non-border column towards South (Procedure `ExploreLine(South)`, followed by Procedure `TowerWalk(East, North)`). The tower was blocked once and that time temporarily ended up in the previously traversed column. (f) A tower traversing the East border column towards North (Procedure `ExploreLine(North)`). The tower was blocked once and that time temporarily ended up in the previously traversed column.

ii) Dir *is North and* u *is on the East border:* v *or West of* v

iii) Dir *is South and* u *is not on the West border:* v *or West of* v

iv) Dir *is South and* u *is on the West border:* v *or East of* v

v) Dir *is West and* u *is on the South border:* v *or North of* v

Proof. Consider a tower of at least three agents at a node $u = (x, y)$ not on the East border. Assume that the agents associated with the tower execute Procedure `ExploreLine` towards direction North (i.e., case (i)). Let v be the node on the North border which is the furthest node on the same column and north of u and let p be the shortest path from u to v. According to Algorithm ExploreLine direction $X = North$ and direction $Y = East$. The tower will execute Procedure `TowerWalk(X, Y)` and by Lemma 3.6 will either reach node $(x + 1, y)$ (which is the node next of u on path p) or node $(x + 1, y + 1)$ (which is the node at distance 2 from u towards direction $X - Y$ (i.e., North-East) within at most 6 time units. The tower agents repeatedly execute Procedure `TowerWalk(X, Y)` from their current node (say u') until they either reach node v in the X (i.e., north) border (in which case the procedure ends and $Parity = 0$ is returned), or they reach a node w which is not straight in direction X (i.e., north) from u for the first time. In that case node w has

to be at distance 2 in direction $X - Y$ (i.e., North-East) of node u'. If w is on the X (i.e., north) border then it must be Y (i.e., East) of v and the procedure ends and $Parity = 1$ is returned. If w is not on the X border, then the agents execute `TowerWalk(Z, X)`, where Z is opposite than Y (i.e., Z is West when Y is East). Therefore the agents will either reach the node next of u' on p or the second next node of u' on p. Hence the agents will either reach node v or the node Y (i.e., east) of v on the X (i.e., north) border within $O(p)$ time units.

If the starting node u is on the East border (case (ii)), then the tower agents always execute `TowerWalk(North, West)`. The proof of this case can be done in a similar way as before by taking $X = North$ and $Y = West$.

If the direction is South and the starting node u is not on the west border (case (iii)), then the proof can be done in a similar way as before by taking $X = South$ and $Y = West$.

If the direction is South and the starting node u is on the west border (case (iv)), then the proof can be done in a similar way as before by taking $X = South$ and $Y = East$.

Finally, if the direction is West and the starting node u is on the south border (case (v)), then the proof can be done in a similar way as before by taking $X = West$ and $Y = North$. □

We now present Algorithm `GridWalk` which gathers $k > 3$ agents in an oriented grid with one malicious agent. The algorithm first instructs the agents to form at least one tower (using Procedure FormTower). Each tower moves to the South-West corner of the grid (using Procedure ExploreLine). Then each tower explores the grid, starting from the South-West corner, and traversing the grid moving back and forth on each column and changing columns from the south border using Procedures ExploreLine and TowerWalk, trying to reach the North-East corner of the grid. Along the way the towers collect any agents which do not belong to any tower. The formal algorithm is shown in Algorithm 3.

Theorem 3.2. *Consider $k \geq 4$ agents in a $n \times m$ oriented grid, initially placed at distinct nodes, with a malicious agent, where $n, m > 1$. If the (honest) agents execute Algorithm* `GridWalk` *then they gather within at most $O(nm)$ time units.*

Algorithm 3: GridWalk

```
/* Algorithm for Gathering k ≥ 4 agents in Oriented Grid                    */
/* State = Tower                                                            */
execute FormTower
if there is an edge towards South then execute ExploreLine(South)
if there is an edge towards West then execute ExploreLine(West)
/* The tower has reached the West border                                    */
while there is an edge towards East do
    execute ExploreLine(North)
    if ExploreLine returns 1 then execute TowerWalk(West, South)
    Perform ExploreLine(South)
    if ExploreLine returns 0 then execute TowerWalk(East, North)
    if ExploreLine returns 1 and the ExploreLine was not performed at the West border
     then
        Let u be the current node
        execute TowerWalk(East, North)
        if tower ended up at distance 2 North-East of u then
            Let v be the current node
            execute TowerWalk(South, East)
            if tower ended up South of v then execute TowerWalk(East, North)
        else execute TowerWalk(East, North)
/* The tower has reached the East border                                    */
execute ExploreLine(North)
if ExploreLine returns 1 then
    while there is an edge towards East do move East
```

Proof. The agents first execute Procedure FormTower. In view of Lemma 3.5, they form one or more groups (towers) of at least 3 agents within $O(n+m)$ time units. Then, each tower first moves to the South and then to the West border (using Procedure ExploreLine). Hence, in view of Lemma 3.7 each tower will eventually reach either exactly the South-West corner of the border, or one node above the South-West corner, of the grid within another $O(n+m)$ time units. Then each tower traverses the whole grid as follows: it traverses each column back and forth (using Procedure ExploreLine) and moves to the next column in direction East (using Procedure TowerWalk). Examples of such traversals are shown in Figures 3.7, 3.8. By repeatedly applying Lemmas 3.6 and 3.7, we get that each tower will either reach the North-East corner or its neighbour node z to the west, within at most $O(nm)$ time units in total. Any tower that reaches node z tries for ever to move East to the North-East corner. Therefore within at most $O(nm)$ time units all towers will meet either at the North-East corner or at node z.

What remains to be proved is that any agents not belonging to a tower will be discovered and collected by a tower on its way to the North-East corner. Notice that any agent H which has not yet been associated with a tower, must still execute Procedure `FormTower` and therefore according to that procedure, agent H either waits at the North-East corner or otherwise tries to move towards North, East, or West[5].

Let us closely follow the exploration of a tower starting at the South-West corner of the grid (or its neighbour node on the north).

Let u be the starting node of such an exploration (i.e., either the South-West corner of the grid or its neighbour node on the north). First notice that if u is not the South-West corner, then it means that the tower could not reach the South-West corner during the execution of Procedure `ExploreLine(`*West*`)`. The only case where this can happen is if the South-West corner is blocked during step t of the last call of Procedure `TowerWalk`, as it can be verified in the procedure. Since agent H cannot move to the South-West corner from any other node, it means that if u is not the South-West corner, agent H can not move at the South-West corner at time t or later.

Hence the tower executes Procedure `ExploreLine(`*North*`)` starting at node u at time t' (e.g., see Figure 3.7(a), (b)). Agent H cannot move to the South-West corner or to node u at t' (without meeting a tower agent) or later. Let v be the node where the tower ends-up after the execution of the procedure. Since $n, m > 1$, this first column is not the East border of the grid.

Suppose that the column is not adjacent to the East border of the grid. In view of Lemma 3.7, since the column (where u is located) is not on the East border, node v is on the North border and either on the same column with u or on the next column towards east. Let t be the first time when one of the tower agents was blocked while trying to move towards North to a node u' on the same column with u (i.e., while executing Procedure `TowerWalk(North, East)` within Procedure `ExploreLine(North)`). Since, up to this moment, no tower agent was blocked while moving North, if agent H was at any node in the shortest path p from u to u', then the tower agents should have met it. Moreover, if u'

[5]We remind the reader that according to Procedure `FormTower`, agent H will try to move West only if it has not yet reached the North border and is located in the East border together with another agent and both of them were blocked in the previous step while trying to go North.

is not adjacent to v then H can not move at u' at time t (without meeting a tower agent) or later since agent H always tries to move towards North or East (unless it is located on the North or East border) and therefore cannot reach u' from a node not in p. If u' is indeed adjacent to v, then it might happen that H will try to move to u' at t or later. However, the tower agents will finish the current Procedure `TowerWalk(North, East)` by either gathering at u' or at a node east of u'. In the first case if H is at v a tower agent will meet H within the first step of the next Procedure `TowerWalk(North, East)`. In the second case if H is at u' or v the tower agents will meet H during the next two executions of Procedure `TowerWalk` (i.e., either during the correction movement by calling Procedure `TowerWalk(West, North)`, or during the next Procedure `TowerWalk(North, East)`). Hence when the tower reaches the north border of the grid at time t', either a tower agent has met H or H will never try to move to a node in the same column with u at time t' or later.

If u, v are not on the same column, then the tower executes Procedure `TowerWalk (West, South)` and moves to a node v' on the same column with u either on the North border or one row below.

Afterwards, the tower executes Procedure `ExploreLine(South)` (e.g., see Figure 3.7(c)) and let w be the node where the tower is located after finishing the execution of Procedure `ExploreLine(South)`.

Since v' was on the West border then in view of Lemma 3.7, nodes v', w either belong to the same column or w is on the next column towards East. If v', w are on the same column, Procedure `ExploreLine` returns 0, and the tower executes Procedure `TowerWalk(East, North)` and moves from w to the next column c towards East at a node v'' either on the South border or one row above (see Figure 3.9). If the tower after `TowerWalk(East, North)` is located one row above the south border, then that means that the malicious agent was blocking the tower from moving to node v'' on the south border, hence there can be no agent H at that location at any later time.

The tower continues exploring this next column c towards north until it reaches the north border at a time t (e.g., see Figure 3.8(d)). If c is not adjacent to the east-border column, then using the same arguments as before, any agent H cannot move to any node of that column c within time t (without meeting a tower agent) or at a later time. Therefore agent

H (if not already met with a tower agent) is located at a column not yet traversed. Then the tower executes Procedure `ExploreLine (South)` and ends up on the south border either on the same column c or one column to the west (e.g., see Figure 3.8(e)). In the first case the tower agent acts exactly as described a few lines above. In the second case, Procedure `ExploreLine` returns 1 and the tower executes 2 or 3 times Procedure `TowerWalk` as it can be seen in Figure 3.10, in order to move to the next (not yet explored) column to the east. Once more if the tower never reaches the initially intended node, then the malicious was blocking the tower (and any other agents) from moving to that node, hence there can be no agent H at that node at that time or any time later.

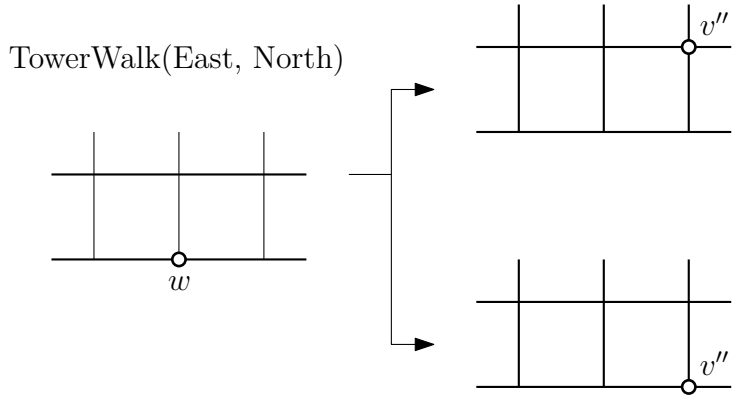

FIGURE 3.9: A tower executed Procedure `ExploreLine(South)` traversing a column and ended up at a node w located in the same column (the procedure returns 0). Then the tower executes Procedure `TowerWalk(East, North)` and ends up at node v''.

Now, consider the case of a tower entering and starting to explore (i.e., towards north) a column c which is adjacent to the East border column of the grid (i.e., c is not the East border but the adjacent column on the west) (e.g., see Figure 3.8(d)). In that case, if agent H is somewhere on the east border, it might happen that H will try to move to a node on the west. Nevertheless it is easy to see that when the tower traverses column c towards North it explores both column c and the east border column. To see why, suppose that agent H is located at a node x in the east border column and let x' be the node on column c which is adjacent to x. Suppose also that the tower is located at a node z in column c on the shortest path between the south border and node x'. Each time the tower executes Procedure `TowerWalk(North, East)`, one tower agent tries to move north of z and another one tries to move east of z. Hence, since agent H will not try moving South, will be either collected by one of the tower agents by the time the tower agents reach the north border

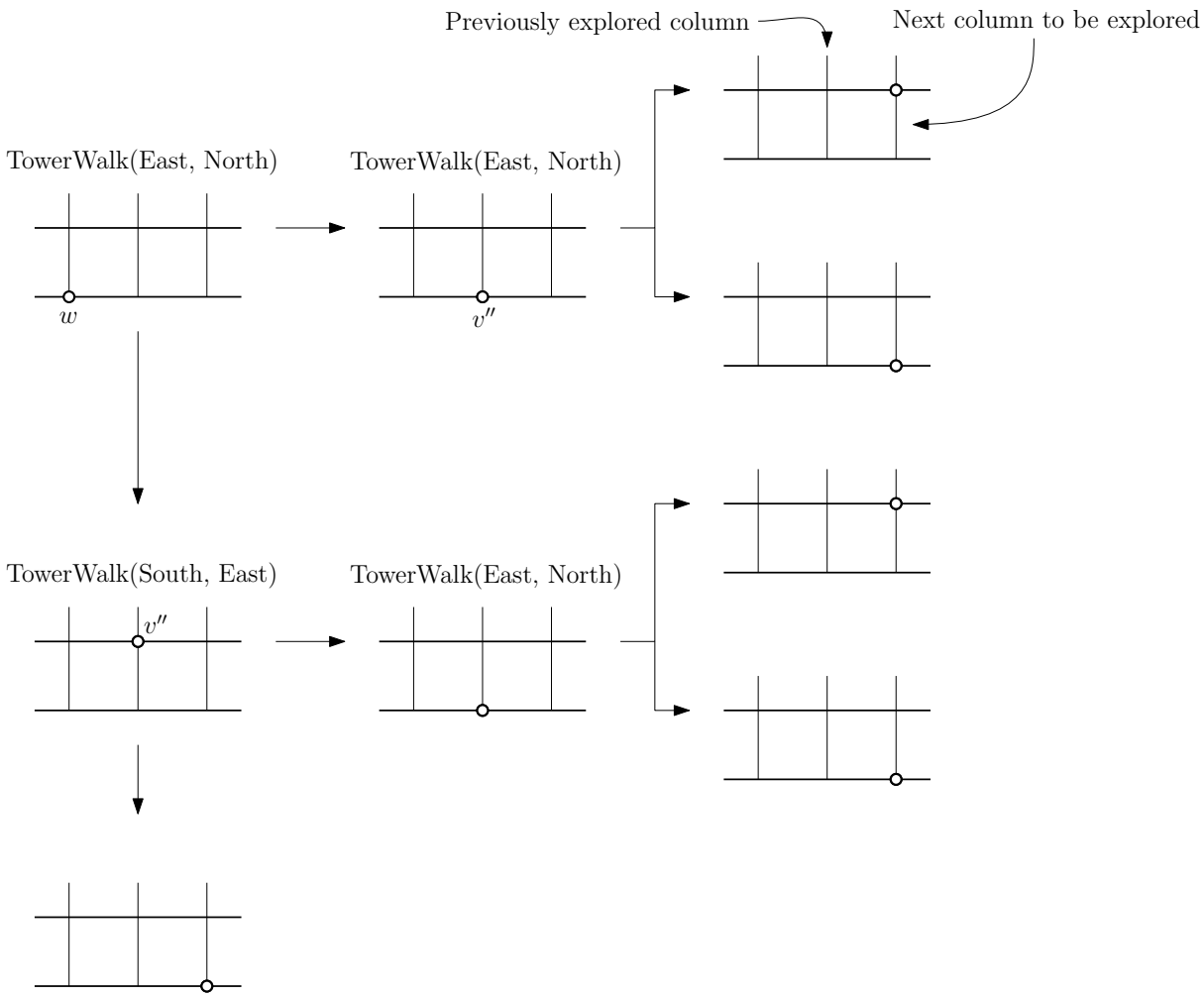

FIGURE 3.10: A tower executed Procedure `ExploreLine(South)` traversing a column which is not on the West border, and ended up at a node w located one column to the West (the procedure returns 1). Then the tower executes Procedure `TowerWalk(East, North)` and ends up at node v''. After that, the tower executes one or two more times the Procedure `TowerWalk` in the appropriate direction(s) and ends up on the next (unexplored) column towards East.

or H will reach the North-East corner of the grid. Notice that by the time the first tower finishes the exploration (i.e., traversing towards north) of the column which is adjacent to the east-border column, if there are agents not associated with a tower, then those agents must wait at the North-East corner of the grid.

Finally, when the tower reaches the East border, the execution of `ExploreLine(`*North*`)` starts, and as a result the tower is now located either at the north-east corner of the grid or at an adjacent node to the west (say y) (e.g., see Figure 3.8(f)). In the second case all agents associated with that tower try to move East until they reach the North-East corner. Notice that, if there is at least one tower at node y trying to go East (at the North-East corner of the grid), then either the malicious agent prevents all agents from moving to the North-East corner, or there is an agent that reaches the North-East corner. Hence all agents will gather either at node y or at the North-East corner. □

3.2.2 Gathering three or more agents

Let us now present a more sophisticated technique for gathering any number of three or more agents. We start with a high-level description of the algorithm.

Notice that in the algorithm for 4 agents, as discussed in Section 3.2.1, the idea was to form a group of 3 agents which can explore the grid (while the members of the group stay close to each other), and this can be done independently of whether other agents are also moving at the same time or not. In the algorithm for 3 agents the idea is to form a group of two agents which can explore the grid (while the members of the group stay close to each other). However now it is necessary that at least one other agent is also trying to explore the whole grid. The reason is that if all other agents plan to move within a limited area, then the adversary could block one of the group agents and therefore it would be impossible for the group agents to explore the whole grid staying close to each other.

For all the reasons mentioned above, the algorithms, procedures, and proofs for 3 agents are much more complicated: A tower of 3 agents (as in the algorithm for 4 agents) can never be blocked by the adversary and it will always explore the grid (even if there were

no other agents). However, a tower of 2 agents (as in the algorithm for 3 agents) can sometimes be blocked without making any progress, but the malicious agent while blocking the tower should allow the alone agents to move freely. Therefore, the algorithms for 3 agents have to be carefully designed so that the tower can not be blocked together with another agent at the same time, which would bring as to a standstill.

The agents executing the algorithm are moving as follows: At least two of the agents meet and form a group of explorers. Then they explore the whole grid moving in a "cautious way" and maintaining a distance of at most one edge between them until they either meet a third agent or they are blocked by the malicious agent. If the *routing distance* h (i.e., the distance with respect to the routing, that we formally define later) between one of the agents in the group and an agent not in the group is $h \leq 2$, then they all meet after a constant number of steps. On the other hand, if the routing distance is $h > 2$, then within $O(h)$ steps either the two agents of the group meet again and their routing distance from the third agent is at most 2, or one of the agents in the group meets the third agent. In this last case all agents meet after a constant number of extra steps. We describe below the algorithm in detail.

Each agent is initially located at a distinct node, and starts at state `initial` moving towards the North border of the grid and then towards the East border until it either reaches the North-East corner or it meets another agent.

If an agent reaches the North-East corner of the grid (i.e., without meeting another agent), it switches to state `Explorer1` and starts exploring the grid by a zig-zag technique: it first traverses the current column to the South; then it proceeds (via an edge on the South border) to the next column to the West and it repeatedly traverses that column initially moving North and then back South until it reaches the South border; it then moves to the next column to the West, and so on (see Figure 3.11). The agent follows the described route with the following additional local moves (see Figure 3.11): it moves one step forward along the route, then back (i.e., to the previous position) and forward again. If the agent is blocked while taking the move backwards it continues moving forward, while if it gets blocked while moving forward it keeps trying to move forward. Hence, if an agent is blocked while moving, on the next step it tries to move again forward until it succeeds.

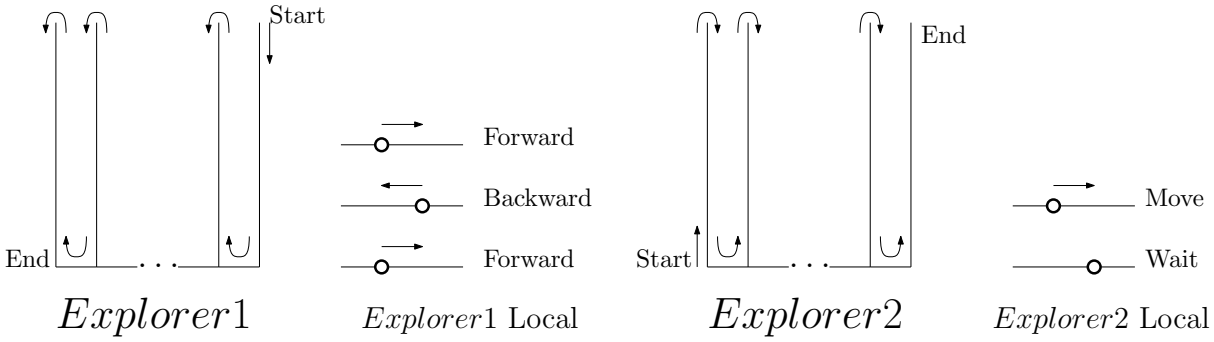

FIGURE 3.11: Both `Explorer1` and `Explorer2` routes are shown. The explorers' local moves depend on the general route direction. For example, when `Explorer1` moves towards South, North, or West, then "forward" means towards South, North or West, respectively.

After having traversed back and forth the column on the Western border of the grid (i.e., reaching for the second time the South-West corner), the agent switches to state `Explorer2` and explores the grid in the opposite way (i.e., traversing one by one the columns from the West to the East and changing columns only via edges on the South border), until it again reaches the North-East corner (see Figure 3.11). The agent at state `Explorer2` follows the described route with the additional local moves: it repeatedly moves one step along the route and then (if not blocked) waits for one time unit before it moves again (see Figure 3.11). In case the agent is blocked while trying to move it tries to move again (i.e., without waiting) until it succeeds.

Each agent repeats those procedures until it meets another agent or reaches the North-East corner of the grid while at state `Explorer2`. We will later show that at least two agents will meet before an agent visits twice the North-East corner of the grid (i.e., at state `Explorer2`). In fact, we will show that at most one agent will switch to state `Explorer2` and before the first agent at state `Explorer2` finishes its traversals, at least two agents meet.

If two agents meet (at any time during the main algorithm) they change their state to `Group-Explorer1` and `Group-Explorer2` respectively (they can always switch to different states since they either come from different directions or one of them did not move in the previous step), and they start (or continue) the exploration in the zig-zag route: the agent at state `Group-Explorer1` follows a similar movement as an `Explorer2` (i.e., starts or continues a zig-zag route towards the North-East corner by repeatedly moving one step forward and waiting for one time unit), while the agent at state `Group-Explorer2`

just follows `Group-Explorer1` with one time unit delay; in other words `Group-Explorer1` moves forward while `Group-Explorer2` waits and on the next time unit `Group-Explorer1` waits until `Group-Explorer2` joins.

The group of explorers moves as described until one of them meets a third agent or `Group-Explorer1` is blocked (notice that only a `Group-Explorer1` could be blocked) by the malicious agent. If they reach the North-East corner they switch direction and move in a similar way towards the South-West corner and then back to the North-East corner.

If `Group-Explorer1` is prevented to move by the malicious agent, then the two agents change their states to `Group-Trace1` and `Group-Trace2` and start moving in a special local way which is described in detail later, which guarantees that they progress in the exploration while maintaining a close distance between them and almost preserve the exploration route. We show later that before the two agents reach twice the North-East corner of the grid they meet a third agent.

Algorithm 4: Gathering $k \geq 3$ agents in a $n \times m$ grid

```
1 State = Initial
2 while you are alone and not on the North border do move North
3 while you are alone and not on the East border do move East
4 if you are alone then
      /* you have reached the North-East corner                    */
5     State = Explorer1
6 else
7     if you have met with exactly one more agent B then
8         if agent B's state is either Initial, Explorer1 or Explorer2 then GroupMove
9         else State = Group-Follower
```

Lemma 3.8. *Consider $k \geq 3$ agents initially placed at distinct nodes in a $n \times m$ grid. By executing Algorithm 4, within $O(n+m)$ time units either at least two agents meet or at least one agent reaches the North-East corner. If $k > 3$ then at least two agents meet. Furthermore, all agents (if any) that meet at the same node switch to distinct states.*

Procedure GroupMove

Let u be the current node
Let $\mathcal{A}$ be the ordered set $\mathcal{A} = \{\texttt{West}, \texttt{North}, \texttt{East}, \texttt{South}\}$
Let d_1 be the direction from which you entered node u
Let d_2 be the direction from which the other agent entered node u
if $d_1 \neq d_2$ **then**
    if d_1 *appears before* d_2 *in set* $\mathcal{A}$ **then** State = `Group-Explorer1`
    else State = `Group-Explorer2`
else
    if *you were already at node* u *in the previous step* **then** State = `Group-Explorer1`
    else State = `Group-Explorer2`

Proof. Since the malicious agent may only block one node u at a time, it might only (forever) prevent agents of the same column to reach the North border. Hence either at least $k-1$ agents finish the execution of the first `while-loop` of Algorithm 4 and reach the North border or at least 2 agents meet within $O(n+m)$ time units. Therefore, if no two agents meet sooner then at least $k-1 \geq 2$ agents will reach the North border within $O(n+m)$ time units.

The malicious agent can divide those $k-1$ agents on the North border, into at most two groups and can prevent the western group from approaching the East border. Notice that, if $k > 3$ at least one of these groups should have at least two agents.

Therefore if $k > 3$ then at least two agents meet while if $k = 3$ either at least two agents meet or at least one agent will reach the North-East corner of the grid within $O(n+m)$ time units.

Moreover, two agents that meet at the same node anywhere on the grid, should have been at different nodes at the previous step and hence they can switch to different states as shown in Procedure `GroupMove`. □

Hence by executing Algorithm 4 within $O(n+m)$ time units either at least two agents have met or one agent has reached the North-East corner of the grid. We show below that within a total number of $O(nm)$ traversals at least two of the $k \geq 3$ agents have met.

Before going on let us present the notation we use in the following proofs and figures. On the figures the transition between configurations are shown. On each configuration the

Procedure Explorer1

```
/* Explorer1 state                                                    */
1  while you are alone and not on the South border do
2      MoveFBF(South)
3  while you are alone and not at the South-West corner do
4      MoveFBF(West)
5      while you are alone and not on the North border do
6          MoveFBF(North)
7      while you are alone and not on the South border do
8          MoveFBF(South)
9  if you are alone then
       /* you have reached the South-West corner                      */
10     State = Explorer2
11 else
12     if you have met with exactly one more agent B then
13         if agent B's state is either Initial, Explorer1 or Explorer2 then
14             GroupMove
15         else State = Group-Follower
```

Procedure MoveFBF(*dir*)

```
/* Move forward-back-forward towards direction dir                    */
1  Move one step at direction dir until you succeed
2  Move one step back
3  if succeeded moving backwards then
4      Move one step at direction dir until you succeed
```

geometric shapes represent the positions of the honest agents, while the arrows denote the move of each corresponding agent. The position of the malicious agent is omitted.

Depending on all possible positions of the malicious agent, two or more configurations may arise. In most cases when the goal of a procedure is achieved the word "Done" followed by a specific time unit t is written on the configuration. Some configurations can occur in two different times from the start of this procedure. We denote $t = \{x, y\}$ if a configuration may occur either at $t = x$ or at $t = y$. When two arrows are associated with an agent, then that agent tries those moves interchangeably until another agent arrives or one of those moves succeeds.

The figures show a very specific movement, usually North-to-South and South-to-North, but the protocol for these moves, and more specifically for the agents in a state of `Group--Trace1` or `Group-Trace2`, can be generalized with a main and a secondary directions. Let us define these directions as (fwd, aux) which take values based on the following rule.

$$\begin{cases} aux \in \{east, west\} & \text{if } fwd \in \{north, south\} \\ aux \in \{north, south\} & \text{if } fwd \in \{east, west\} \end{cases}$$

Lemma 3.9. *Consider $k \geq 3$ agents initially placed at distinct nodes in a $n \times m$ grid. By executing Algorithm 4, within $O(nm)$ steps at least two agents meet. Furthermore, all agents that meet at the same node switch to distinct states.*

Proof. In view of Lemma 3.8, if $k > 3$ then at least two agents meet while if $k = 3$ either at least two agents meet, or at least one agent has reached the North-East corner of the grid within $O(n + m)$ time units.

Suppose that $k = 3$ and no two agents have met until one agent reaches the North-East corner of the grid. Let A be the first agent that reaches the North-East corner. According to Algorithm 4, agent A's state changes to `Explorer1` and then moves according to Algorithm Explorer1. Agent A will either meet another agent as it moves, or it will be blocked, possibly for many time units, or it will reach the South-West corner of the grid without meeting an agent. It is easy to see that agent A can not be overtaken by another agent at state `Explorer1` without the two agents meet. Therefore, as long as no agents have met,

all other agents at state `Explorer1` except agent A, may only exist in the same column as A or in different columns closer to the East border of the grid.

Suppose that, while no agents have met, agent A is blocked while moving forward[6] towards a node u (i.e., the malicious agent is at u). There are three cases for the location of node u. It could be either on the north border, on the south border, or somewhere in between.

- If node u is on the North border, then only an agent (at state `Initial`) coming from the West could be blocked at the same time, since no agent would try to move to node u coming from the East. Hence, in that case, at most one more agent could be blocked while moving to u.
- If node u is not on the North or the South border, then only an agent coming from the opposite direction of A (i.e., moving in the same column as A) could be blocked at the same time, since no agent would try to move to node u coming from the East or West. Hence, again at most one more agent B could be blocked while moving to u (notice that B should either be at state `initial` if u is at the East border, or at state `Explorer1`, otherwise).
- If node u is on the South border, then again only one more agent B could be blocked at the same time: when A is coming from the North and B is coming from the East (notice that B should also be at state `Explorer1`).

Since, in all above cases, at most two agents can be blocked at the same time, either A will be unblocked, or a third agent will meet agent A or B, within $O(nm)$ time units. Hence either two agents meet or agent A is the first agent that reaches the South-West corner, switches to state `Explorer2` and moves according to Algorithm Explorer2.

We now show that at least two agents meet before agent A reaches again the North-East corner. To show this we first observe that while no two agents have met and A has not been blocked, if at a time t there is an agent at a node v and at state `Initial` while agent A (at state `Explorer2`) is moving towards the North-East corner, then agent A will pass from v at a time $t' > t$ and before reaching the North-East corner. We also observe that as

[6] We analyze only the forward moves of agents at state `Explorer1`, since if such an agent is blocked while moving backwards, it immediately tries to move forward until it succeeds. Hence it can not be blocked for more than one time unit trying to move backwards.

long as no two agents have met and agent A has not been blocked: if at a time t there is an agent B at a node v and at state `Explorer1` while agent A (at state `Explorer2`) is moving towards the North-East corner, then agent A would either pass from v at a time $t' > t$ and before reaching the North-East corner or agents A, B, will meet at time $t + 1$.

Hence while agent A at state `Explorer2` is moving towards the North-East corner without having been blocked and before two agents meet, any other agent B is either at state `Initial` or `Explorer1`. Moreover as long as no two agents have met and agent A is not blocked, A will meet with another agent B before reaching the North-East corner.

We now show that if A is blocked, then it will either eventually be unblocked or otherwise two agents will meet after a finite number of time units. If A is blocked while trying to move to a node u then it might happen that another agent B is also blocked at the same time while trying to move to u (in that case agent B cannot be at state `Initial`). Similarly as in the cases above (i.e., when agent A was at state `Explorer1`), if node u is not on the South border, then no third agent can be blocked while moving to u[7], and since any other agent at state `Explorer1` should pass from node u on its way to the South-West corner either A will be unblocked or a third agent would meet agent A or B within $O(nm)$ time units. If however node u is on the South border, then there is a single case where 3 agents could be blocked at the same time: it might happen that B is at state `Explorer1` coming from the North and moving to u and another agent C at state `Explorer1` is coming from the East and moving to u while A was trying to move to u coming from the West. Note that there can be no two agents at state `Explorer2` since if agent B was at state `Explorer2` it would have met agent A.

In order to avoid a scenario like this, agent A (at state `Explorer2`) is instructed to deviate a little from its usual algorithm when it is blocked while trying to move to a node on the South border, coming from the West, on its way to the North-East corner, while agents B, C execute Algorithm Explorer1. The details are shown in Figures 3.12, 3.14 where all possible scenarios are demonstrated.

Because of this route-deviation of agent A, now it is not clear whether agents A, B could

[7] Actually when A was previously blocked on the South border it then deviates for a few moves from its usual algorithm (see this deviation of A later in the proof), and it might happen that three agents are blocked at the same time. However, in that case, as we will see, agent A will soon be unblocked.

miss each other. However, we show in Figures 3.12, 3.14 that in any configuration A, B cannot miss each other and continue their way to the corners of the grid.

Hence in all cases at least two agents will meet before agent A reaches for the second time the North-East corner of the grid. Two agents that meet at the same node anywhere on the grid, they should have been in different configurations at the previous step and hence they can switch to different states as shown in Procedure `GroupMove`. □

Procedure Explorer2

```
/* Explorer2 state                                                    */
1  MoveFW(East)
2  while you are alone and not on the North border do
3      MoveFW(North)
4  while you are alone and not at the North-East corner do
5      while you are alone and not on the South border do
6          MoveFW(South)
7      if you are alone then MoveFW(East)
8
9      while you are alone and not on the North border do
10         MoveFW(North)
11 if you have met with exactly one more agent B then
12     if agent B's state is either Initial, Explorer1 or Explorer2 then
13         GroupMove
14     else State = Group-Follower
15
```

Procedure MoveFW(*dir*)

```
/* Move forward-wait towards direction dir                            */
1  Move one step at direction dir
2  if you have been blocked then
3      if (dir = East) AND (state = Explorer2) then
4          Move as shown in Figures 3.12, 3.14
5      else
6          Move one step at direction dir until you succeed
7          Wait for one time unit
8  else
9      Wait for one time unit
```

Notice that, all agents except those at state `Initial` are moving on the same route (excluding a temporary deviation of an `Explorer2` when it is blocked on the South border).

Procedure Group-Explorer1

```
/* Group-Explorer1 state                                          */
/* First moving towards the North-East corner                     */
Group-Explorer1MoveNE
/* Then moving towards the South-West corner                      */
Group-Explorer1MoveSW
/* Then moving again towards the North-East corner                */
Group-Explorer1MoveNE
if you have been blocked and you have not met a third agent then
    State=Group-Trace1
```

Procedure Group-Explorer1MoveNE

```
/* Moving towards the North-East corner                           */
while you have not met a third agent and you are not on the North border and not
 blocked do
    MoveFW(North)
while you have not met a third agent and you are not at the North-East corner and
 not blocked do
    while you have not met a third agent and you are not on the South border and not
     blocked do
        MoveFW(South)
    if you have not met a third agent and not blocked then MoveFW(East)
    while you have not met a third agent and you are not on the North border and not
     blocked do
        MoveFW(North)
```

Procedure Group-Explorer1MoveSW

```
/* Moving towards the South-West corner                           */
while you have not met a third agent and you are not on the South border and not
 blocked do
    while you have not met a third agent and you are not on the South border and not
     blocked do
        MoveFW(South)
    if you have not met a third agent and not blocked then MoveFW(West)
    while you have not met a third agent and you are not on the North border and not
     blocked do
        MoveFW(North)
while you have not met a third agent and you are not at the South-West corner and
 not blocked do
    MoveFW(South)
```

Procedure Group-Explorer2

```
/* Group-Explorer2 state                                                    */
Wait for one time unit
/* First moving towards the North-East corner                               */
Group-Explorer2MoveNE
/* Then moving towards the South-West corner                                */
Group-Explorer2MoveSW
/* Then moving again towards the North-East corner                          */
Group-Explorer2MoveNE
if you have been blocked and you are together only with a Group-Explorer1 then
    State=Group-Trace2
```

Procedure Group-Explorer2MoveNE

```
/* Moving towards the North-East corner                                     */
while you have not met a third agent and you are not together with a
 Group-Explorer1 on the North border do
    MoveFW(North)
while you have not met a third agent and you are not together with a
 Group-Explorer1 at the North-East corner do
    while you have not met a third agent and you are not together with a
     Group-Explorer1 on the South border do
        MoveFW(South)
    if you have not met a third agent and you are not together with a Group-Explorer1
     then
        MoveFW(East)
    while you have not met a third agent and you are not together with a
     Group-Explorer1 on the North border do
        MoveFW(North)
```

Procedure Group-Explorer2MoveSW

```
/* Moving towards the South-West corner                                     */
while you have not met a third agent and you are not together with a
 Group-Explorer1 on the South border do
    while you have not met a third agent and you are not together with a
     Group-Explorer1 on the South border do
        MoveFW(South)
    if you have not met a third agent and you are not together with a Group-Explorer1
     then
        MoveFW(West)
    while you have not met a third agent and you are not together with a
     Group-Explorer1 on the North border do
        MoveFW(North)
while you have not met a third agent and you are not together with a
 Group-Explorer1 at the South-West corner do
    MoveFW(South)
```

Let us now define the *routing distance*, denoted by $h_t(A, B)$, between agents A and B at time t:

Definition 3.3. *Let A be a group agent colocated with its related group agent and let B be either an agent at state* `Explorer1`, `Explorer2` *or another group agent colocated with its related group agent. Agents A and B are executing the corresponding algorithms and hence they are moving on the same main route between the North-East and the South-West corner back and forth as instructed by the algorithms. Suppose that agent A occupies node u and agent B occupies node v at time t. We define the* routing distance $h_t(A, B)$ *as the number of edge traversals, without including back and forth or local moves, the agents need to do (where each agent follows its instructed route) in order to meet if the agents had not been blocked.*

In Figure 3.18 the *routing distances* of agents A, B are shown in three different cases.

The procedures that are executed by an agent at state `Group-Trace1` and the other agent at state `Group-Trace2` are quite technical and therefore are only shown in the corresponding figures. An agent at state `Group-Follower` "sticks" together with (i.e., follows) the agent that met.

In view of Lemma 3.9, within $O(mn)$ time units there is at least one group of agents and any agent not belonging to a group is either at state `Initial`, `Explorer1` or `Explorer2`.

Observe that, if an agent at state `Explorer1` is blocked while trying to move forward or backward then it tries to move forward on its route (i.e., at most every 2 time units tries to move forward). Also observe that, an agent at state `Explorer2` tries to move forward on its route every 2 time units (unless it was blocked on the South border going East).

In the next lemmas we analyze the movements of agents at the explorer states (i.e. `Explorer1` and `Explorer2`) without taking into account the local movements (shown in Figure 3.11) of those agents for keeping the presentation simpler. Nevertheless the results continue to hold even if we include the local movements of the agents, since the only difference is that due to the local movement an agent instead of going forward it might first go back or wait for one step (unless it was blocked on the South border going East, which case we handle separately). In order to explain further why these local moves do not interfere with the algorithms while ensuring the gathering of the two agents moving

in opposite directions let us consider the following. `Explorer2`, which is moving one step forward and then waits for one time unit, is located at any node for at least two time units, while `Explorer1`, which moves one step forward, then one step backward, and one step forward again, is located at a node different from the one trying to reach for at most one time unit. Therefore, there are not enough time units in order for an `Explorer1` and an `Explorer2` to cross on an edge of the grid without noticing each other. They are going to meet at a node, assuming there is no malicious agent interfering.

Lemma 3.10. *Let $k \geq 3$ agents initially placed at distinct nodes in a $n \times m$ grid. Suppose that the agents are executing Algorithm 4. Consider a group of explorers A_1, A_2 at states* `Group-Explorer1` *and* `Group-Explorer2` *respectively, which are co-located at a node u and an alone agent A_3 at state* `Explorer1` *or* `Explorer2` *which is located at a node v at time $t = 0$. Suppose also that all agents are about to start their movement. If the routing distance between the group of agents and the alone agent is $h_0(A_1, A_3) \leq 2$ then within at most 11 additional time units either all three agents meet or the alone agent has met another agent.*

Proof. If $h_0 = 1$, then in the next time unit `Group-Explorer1` and the alone agent will cross the same edge (without noticing each-other) effectively swapping positions, while the alone agent meets `Group-Explorer2`. At the next time unit `Group- Explorer2` and the previously alone agent move to `Group-Explorer1` while `Group-Explorer1` waits, thus all three agents meet within a total of 2 time units.

If $h_0 = 2$ then the malicious agent can either block or not block the agents by moving to the node (say w) between the group explorers and the alone agent.

If the malicious agent does not block the agents then `Group-Explorer1` will meet with the alone agent at node w. On the next time unit the agent in state `Group-Explorer2` moves to w while both `Group-Explorer1` and the previously alone agent wait. Thus after a total of 2 time units all three agents meet.

In the last case where the malicious agent occupies node w the agents `Group-Explorer1` and `Group-Explorer2` change their states to `Group-Trace1` and `Group-Trace2` respectively and no agent moves. Depending on the exact location of node u there are a number of cases which are shown in Figures 3.15, 3.16, 3.17, 3.19 and 3.20.

If node w is on the South border and the agent at state `Explorer2` was trying to move to w coming from a node West of w on the South border, then the situation is depicted in Figures 3.15, 3.16. A similar case where w is one row above the south border but again the agent at state `Explorer2` is trying to move to w coming from west of w is depicted in Figure 3.17.

In every other case, on the next time unit the agents start a procedure as shown in Figure 3.19. The starting configuration for this case along with all resulting configurations until all agents meet (if no other agents interfere) are depicted in Figures 3.19 and 3.20. Hence, if no other agents interfere, then the agents meet within at most 11 time units. If within less than 11 time units, a fourth agent interferes and meets one of the group agents, then if the agent was alone, it becomes a follower of the group, while if it was a member of another group then one of them is elected as a follower of the other group. If another agent interferes and meets the alone agent, then the alone agent joins another group.

Notice that the protocols are consistent since for any two possible configurations C_1, C_2 at any fixed time t the strategy of a fixed agent A is the same when the recent history of A is the same at both C_1, C_2.

□

Lemma 3.11. *Let $k \geq 3$ agents in a $n \times m$ grid initially located at distinct nodes. Suppose that the agents are executing Algorithm 4. Consider a group of explorers A_1, A_2 at states* `Group-Trace1` *and* `Group-Trace2` *respectively, which are located at nodes u, v, where $u \neq v$ and an alone agent at state* `Explorer1` *or* `Explorer2` *which is located at node u at time $t = 0$ in the configurations shown in Figure 3.20. Then after at most 4 additional time units, all three agents meet.*

Proof. Agents A_1 and A_2 are located at different nodes either at distance one or at distance two diagonally from each other. In the first case if the alone agent is together with either A_1 or A_2 then the three agents meet on the next time unit as it is shown in Figure 3.19. In the second case where the agents of the group are at distance two diagonally from each other, the alone agent somehow helps the two group agents in order to meet.

The agents of the group behave in the following manner. One agent is waiting, while the other is trying to move either south or west interchangeably until either it succeeds or the group agent meets with another agent that moves to the same node. Notice that these moves depend on the general direction of the group. We suppose that the group was moving south without loss of generality since we can change the direction of the moves based on the rotation and the forward and secondary directions mentioned earlier. Thus, if there is no other agent the two group agents can be blocked by the malicious agent indefinitely. When the alone agent meets the agent that tries to move interchangeably south and west, they can do a simultaneous move (i.e., one agent moves west and the other south) and after at most three time units all three agents gather. If the alone agent meets the group agent that waits, it tries to move in such a way in order to create a "bridge". As shown in Figure 3.20 that move is either north or east (which is actually the opposite moves of south and west). In any case it is easy to see that after at most three time units all agents gather.

The proof is depicted in Figure 3.20. Notice that the protocols are consistent since any two possible configurations C_1, C_2 at any fixed time t the strategy of a fixed agent A is the same when the recent history of A is the same at both C_1, C_2.

□

Lemma 3.12. *Let $k \geq 3$ agents in a $n \times m$ grid, initially located at distinct nodes. Suppose that the agents are executing Algorithm 4. Consider a group of explorers A_1, A_2, at states* `Group-Explorer1` *and* `Group-Explorer2` *respectively, which are co-located at a node u and an alone agent A_3 at state* `Explorer1` *or* `Explorer2` *which is located at a node v at time $t = 0$. Suppose also that all agents are about to start their movements. Let $h_0(A_1, A_3)$ be the routing distance at time $t = 0$ between the group of agents and the alone agent. Then at time $t \leq 8h_0 + 11$ either three agents meet, or the alone agent has met another agent.*

Proof. If $h_0(A_1, A_3) \leq 2$ then in view of Lemma 3.10 all agents meet within at most 11 time units or the alone agent meets another agent. Consider the remaining case where $h_0(A_1, A_3) > 2$. We will first prove that within a time $t^* \leq 8h_0$, if no other agent interferes, then either all three agents meet or one of the group explorers meets the alone agent in the configurations shown in the upper part of Figure 3.20.

Then in view of Lemma 3.11 within an additional time of at most 4 time units all three agents meet.

In order to prove our first claim, we will show that every at most 8 time units there is at least one of the group agents $A' \in \{A_1, A_2\}$ (not necessarily always the same) for which the routing distance between A' and the alone agent A_3 strictly decreases by at least one. Hence eventually one of the group agents meets the alone agent. We will also prove that when this happens, the possible formed configurations are the ones shown in the upper part of Figure 3.20.

As it can be seen in Figure 3.23, within at most $t_1 \leq 4$ time units, either there is at least one of the group agents $A' \in \{A_1, A_2\}$ for which the distance between A' and node v (i.e., the node where the alone agent A_3 was located at time $t = 0$) has strictly decreased by at least one, or the situation is the one depicted at the bottom-rightmost configuration. In the last case after 4 time units more there is at least one of the group agents $A' \in \{A_1, A_2\}$ for which the distance between A' and node v has strictly decreased by at least one.

Notice that, at every single time unit, at least one of the agents A_1, A_2 either is located on the main-route where all three agents move or tries to move there and hence there is no chance that agent A_3 could escape without meeting at least one of the group agents. Furthermore, within those 8 time units, either agent A_3 has moved at least two times, or there is at least one of the group agents A' for which the distance between A' and node v has strictly decreased by at least two. Therefore within a total of at most 8 time units there is at least one of the group agents $A' \in \{A_1, A_2\}$ for which the distance between A' and agent A_3 has strictly decreased by at least one. There is a special case where A_3 is an `Explorer2` moving on the south border trying to move to the same column the group tries to move to, where the routing distance is actually greater than two, but their normal distance is 2 and hence this case is as if their routing distance was 2. Then in view of Lemma 3.10 all agents meet within at most 11 time units or the group passes and progress is made as shown in Figure 3.15.

Hence within a total of at most $8h_0$ time units, at least one of the group agents meets the alone agent. The possible configurations are shown in the upper part of Figure 3.20. Notice that it might be the case that A_3 is collected by another group which was closer.

Then the meeting would not occur but there will be progress in the general sense. □

Lemma 3.13. *Consider $k \geq 3$ agents in a $n \times m$ grid, initially located at distinct nodes. Suppose that the agents are executing Algorithm 4. Then, within $O(nm)$ time units, either at least 3 agents gather or there is no agent at state* `Initial`*.*

Proof. In view of Lemma 3.9, two agents A, B meet at a time $t = O(nm)$. Suppose that at time t the remaining agent C is at state `Initial`. We show below that within $O(mn)$ time units, either at least three agents meet, or all non-group agents have switched to state `Explorer1` or `Explorer2`. In other words, an agent C cannot be blocked 'for a long time' to state `Initial`. Notice that, if an agent C at state `Initial` is not blocked while moving then it will either meet an agent (hence entering a different state) or it will reach the North-East corner of the grid switching to state `Explorer1`.

Suppose that C is not located at the North border and is blocked while trying to move towards the North border. Then the situation is similar to the one that agent C is at a state `Explorer1` or `Explorer2`, and for those cases we proved that C soon meets at least one of the group agents.

Suppose that C is located at the North border and is blocked while trying to move East. Then the situation is similar to the one that agent C is at state `Explorer1` located on the South border and trying to switch columns towards the West border, and for those cases we proved that C soon meets at least one of the group agents.

□

Lemma 3.14. *Consider $k \geq 3$ agents in a $n \times m$ grid initially located at distinct nodes. Suppose that the agents are executing Algorithm 4. Consider a group of explorers A_1, A_2 at states* `Group-Explorer1` *and* `Group-Explorer2` *respectively, which are co-located at a node u and another group of explorers B_1, B_2 at states* `Group-Explorer1` *and* `Group-Explorer2` *respectively, which are co-located at a node v at time $t = 0$. Suppose also that all agents are about to start their movements. If the routing distance between the two groups of agents is $h_0(A_1, B_1) \leq 2$ then within at most 8 time units at least one group of at least 3 co-located agents has been formed.*

Proof. When $h_0 = 1$ the groups cannot be blocked since they are on adjacent nodes moving toward each other. After one time unit the Group-Explorers 1 meet the Group-Explorers 2 of the opposite group. Since the groups where moving opposite to each other, one was moving toward the northeast corner and the other toward the southwest corner. The group that was moving toward the northeast corner follows the other group.

When $h_0 = 2$ and the groups are not blocked, the Group-Explorers 1 of both groups meet after one time unit, then after one more time unit the Group-Explorers 2 will join them. When the groups are blocked they change states into group traces. As it is shown in Figures 3.21 and 3.22 the agents move following the group-trace protocol, except when two agents of different groups meet. As shown in the figures, that is bound to happen after at most three time units.

The case in Figure 3.21 would be the same on the South-border if we rotate every configuration $90°$ counter-clockwise. This is achieved with the following transformation of the fwd and aux directions respectively. For the group moving South the directions change in the following way: fwd from **S** to **E** and aux from **E** to **N**.

Those agents carry now the information that the two groups are near each other. Having that knowledge the agents can now deviate from the original group trace protocol and as shown in the figures, after at most eight time units at least one group of at least 3 co-located agents has been formed. □

Lemma 3.15. *Consider $k \geq 3$ agents in a $n \times m$ grid initially located at distinct nodes. Suppose that the agents are executing Algorithm 4. Consider a group of explorers A_1, A_2 which are co-located at a node u and another group of agents B_1, B_2 which are co-located at a node v at time $t = 0$. Suppose also that all agents are about to start their movements. Let $h_0(A_1, B_1)$ be the routing distance at time $t = 0$ between the two groups of agents. Then at time $t \leq O(h_0)$ at least one group of at least three agents has been formed.*

Proof. At $t = 0$ the initial routing distance between the two groups of explorers is h_0. If $h_0 > 2$ then after a constant number of steps and because the malicious agent can only block one of the two groups, group A and group B, will get closer. Let us consider without loss of generality that group A is the group which is not blocked. Then, group A being free to move will reach, after a constant number of steps, a node which is closer to group B.

Hence, the new routing distance $h_1(A_1, B_1)$ is strictly less than the initial routing distance (i.e. $h_0 < h_1$).

The same procedure goes on for a number of steps $O(h_0)$, until the two groups reach a routing distance $h_l \leq 2$. Then it follows from Lemma 3.14 that the two groups meet. The two groups will meet if and only if they are moving on opposite direction on the same column, or on the south border, or trying to move to the same node. Since the agents move to the north-east corner of the gird and then change their direction moving opposite towards the south-west corner, any two groups will eventually move in opposite directions during the execution of the algorithm. □

When a group of at least three agents is formed, the group can still execute the previous algorithms which consider a group of two agents. The agents split into two groups which represent the two agents which are in state `Group-Explorer1` and `Group-Explorer2` and continue the movement of the group from where it stopped. Hence we have the following theorem.

Theorem 3.3. *Consider $k \geq 3$ agents in a $n \times m$ oriented grid, initially placed at distinct nodes, with a malicious agent, where $n, m > 1$. If the (honest) agents execute Algorithm 4 then they gather within $O(nm)$ time units.*

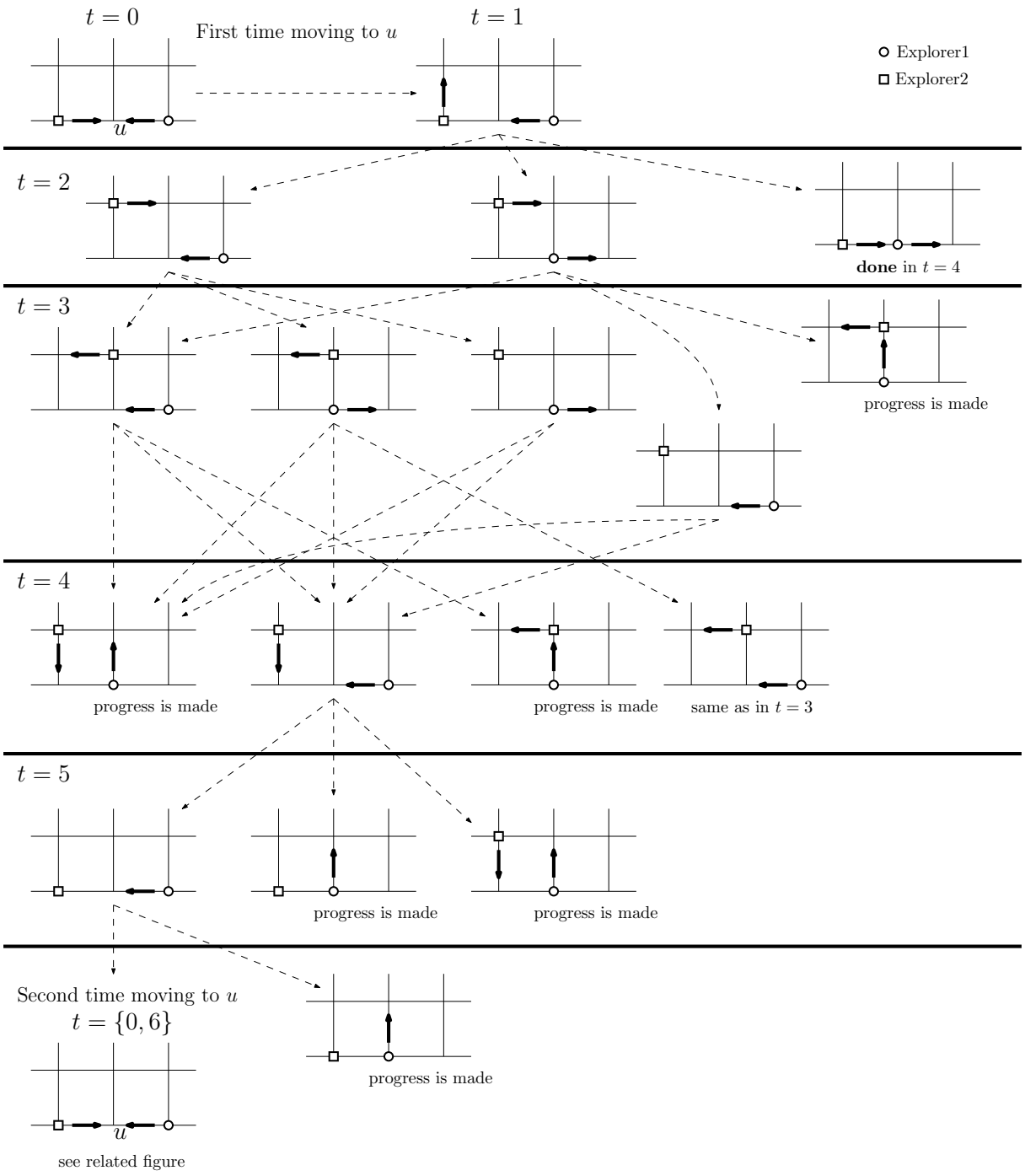

FIGURE 3.12: This is the case where `Explorer2` is coming from the West, whereas `Explorer1` is coming from the East. Both of them want to switch columns and go to node u. We can see that if `Explorer2` follows the depicted protocol, then `Explorer1` will either manage to switch column and therefore progress is done, or a third agent will meet either `Explorer1` or `Explorer2`. Notice that the protocol is consistent since at any given time the moves of the agents are the same for all configuration of that time, unless their very recent history is different. For the second time moving to u see Figure 3.13.

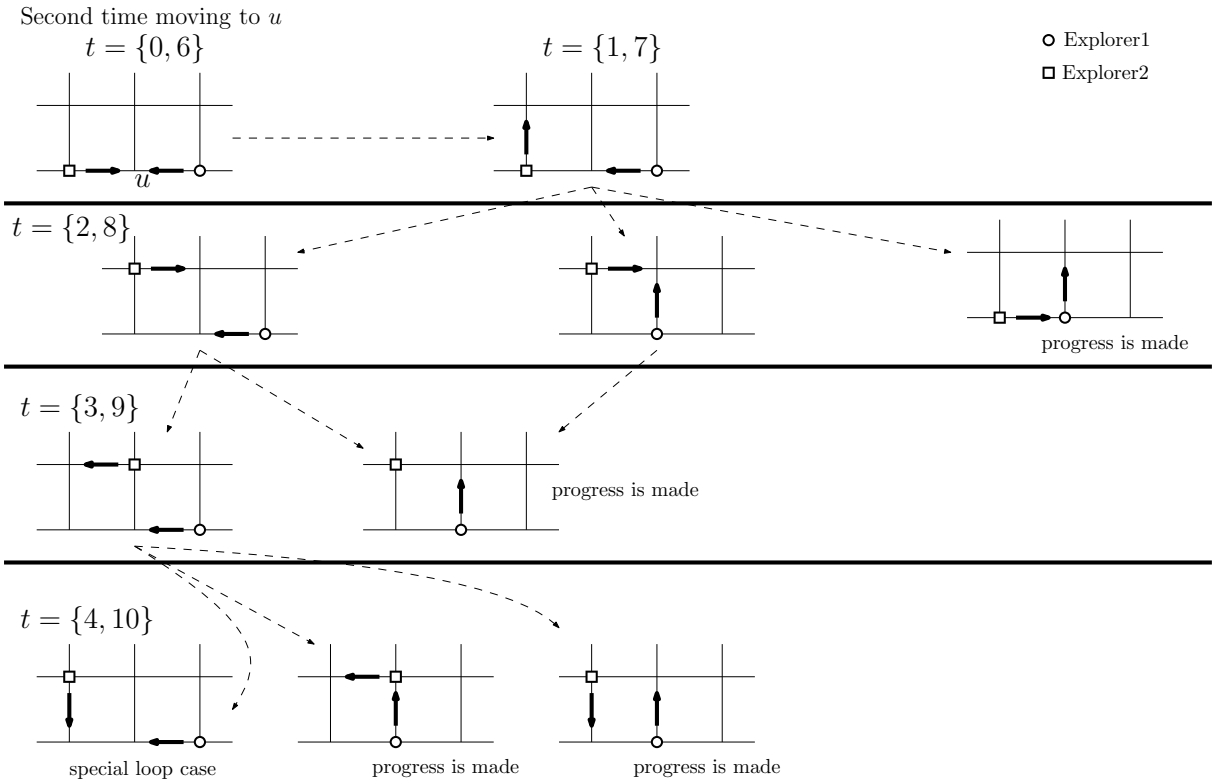

FIGURE 3.13: This is the case where `Explorer2` is coming from the West, whereas `Explorer1` is coming from the East for the second time due to local movement (forward-backward-forward). Both of them want to switch columns and go to node u. We can see that if `Explorer2` follows the depicted protocol, then `Explorer1` will either manage to switch column and therefore progress is done, or a third agent will meet either `Explorer1` or `Explorer2` even if they are blocked in a loop as shown at the bottom leftmost configuration. Notice that the protocol is consistent since at any given time the moves of the agents are the same for all configuration of that time, unless their very recent history is different.

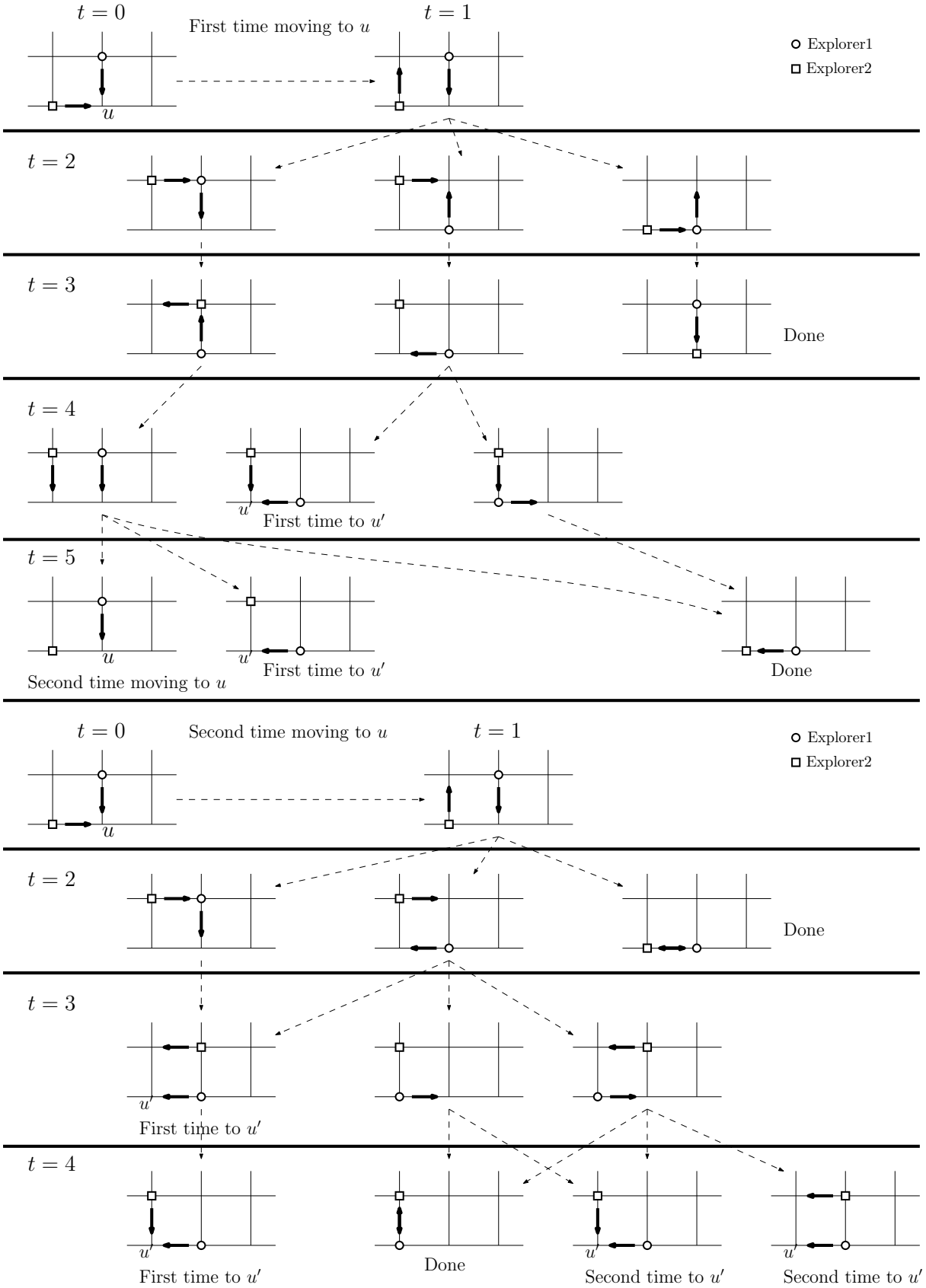

FIGURE 3.14: This is the case where `Explorer2` is coming from the West, whereas `Explorer1` is coming from the North. In $t = 3$ if the West move by `Explorer2` fails then the same move is executed until it succeeds. In $t = 4$ the South move follows the same principle "try South until you succeed". For the cases where `Explorer2` is not on the south border and tries to move south towards the same node as `Explorer1` then the agents can be blocked until a third agent meets with one of them.

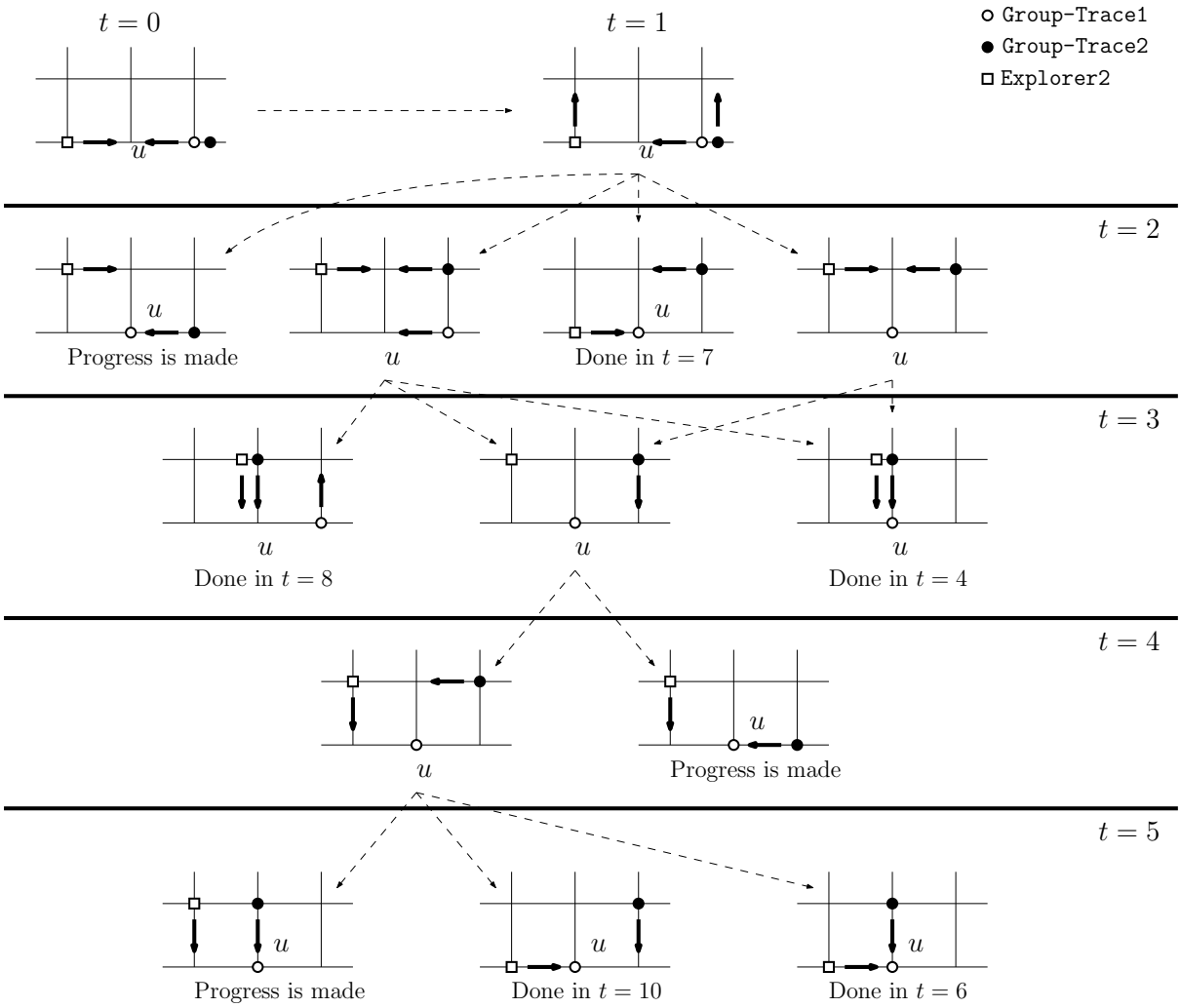

FIGURE 3.15: The case where a group of explorers, which are at states `Group-Explorer1` and `Group-Explorer2`, and an alone agent, which is at state `Explorer2`, are blocked on the South border.

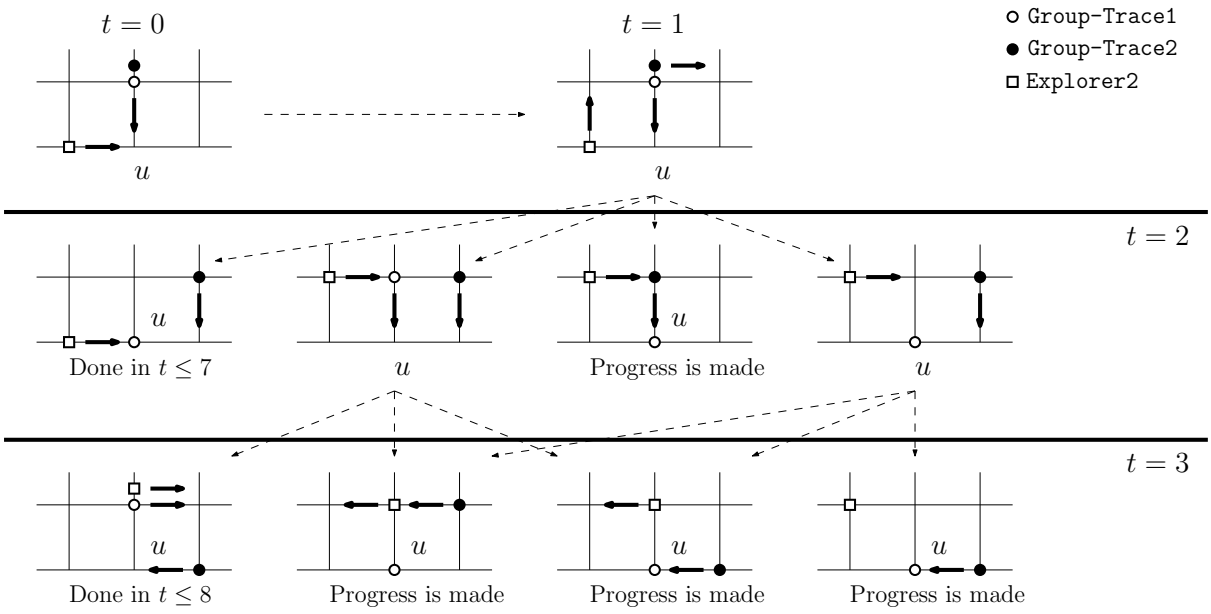

FIGURE 3.16: The case where a group of explorers, which are at states `Group-Explorer1` and `Group-Explorer2`, and an alone agent, which is at state `Explorer2`, are blocked on the South border.

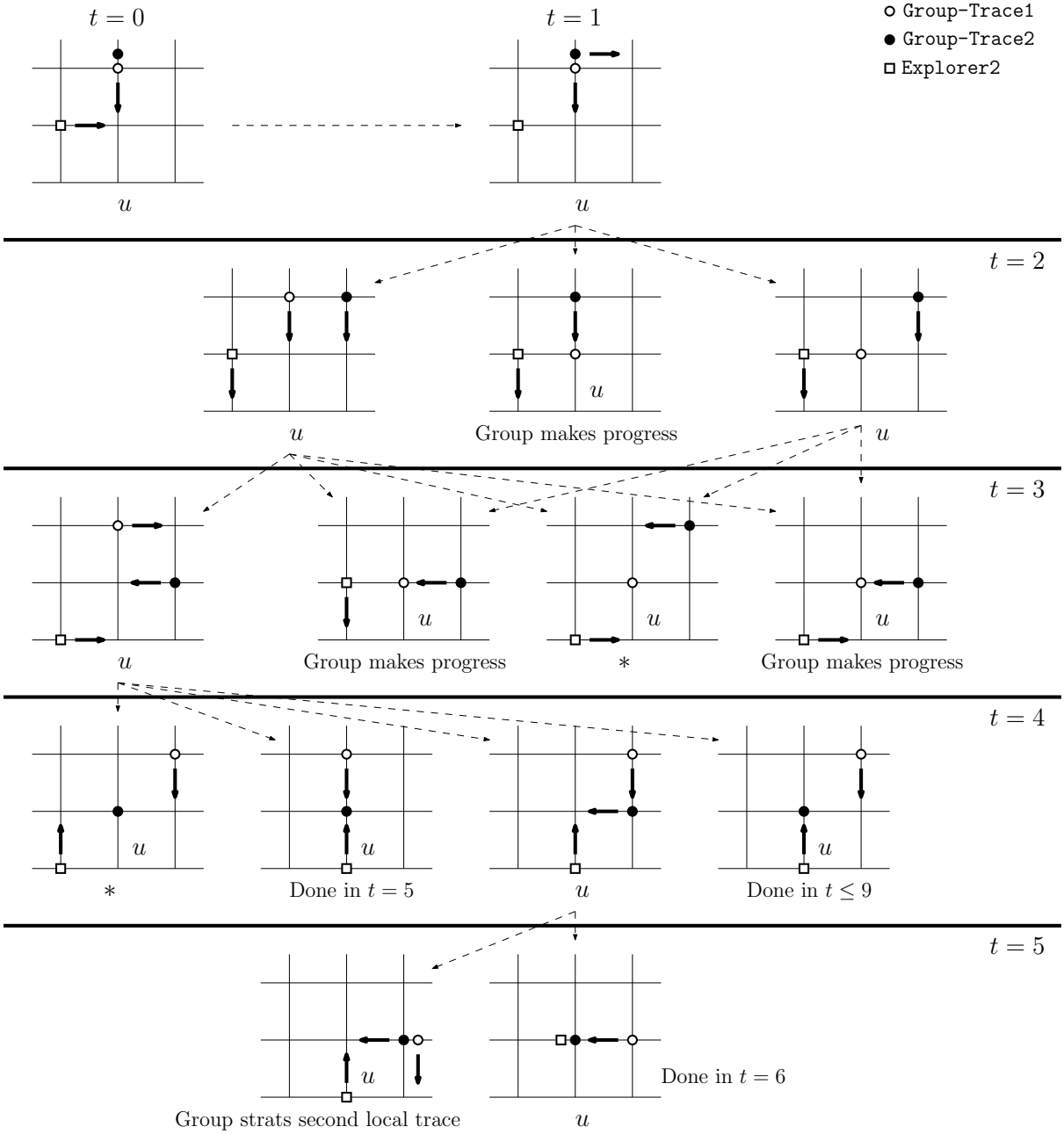

FIGURE 3.17: The case where a group of explorers, which are at states `Group-Explorer1` and `Group-Explorer2`, and an alone agent, which is at state `Explorer2`, are blocked on the South border.

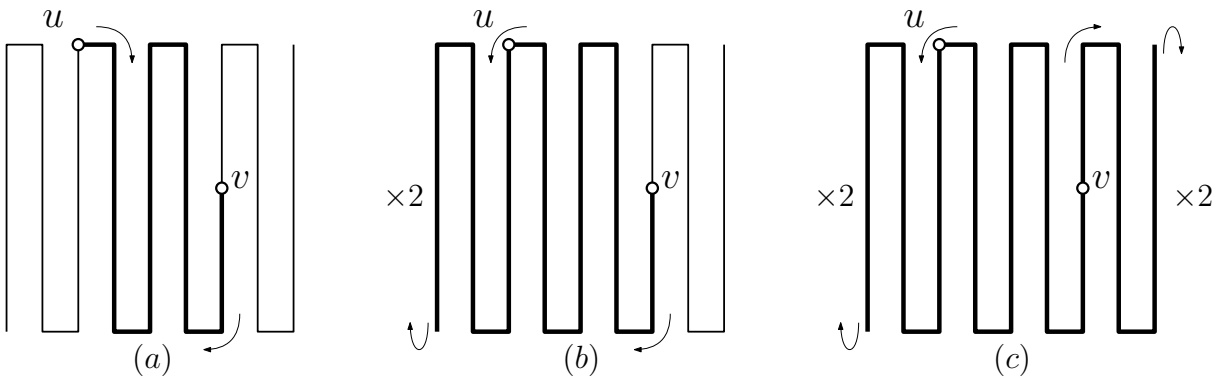

FIGURE 3.18: Let A, B be two agents located at nodes u, v respectively. The routing distance is depicted in three examples. (a) The agent located at u is moving towards the North-East corner by first going to the South border while the agent located at v is moving towards the South-West corner by first going to the South border: in that case the routing distance is the number of edge traversals one of the agents had to do in order to meet the other agent if the other agent was not moving. (b) The agents are moving towards the South-West corner of the grid and both agents are first moving towards the South border: in that case the routing distance is the number of edge traversals the agent at u has to do in order to meet the other agent if the other agent was not moving. (c) The agent located at u is moving towards the South-West corner by first moving towards the South border of the grid, while the agent located at v is moving towards the North-East corner by first moving towards the North border of the grid: in that case the routing distance is the number of edge traversals the agent at u has to do in order to reach node v plus the number of edge traversals the agent at v has to do in order to reach again node v with direction the South-West corner by first going towards the South border.

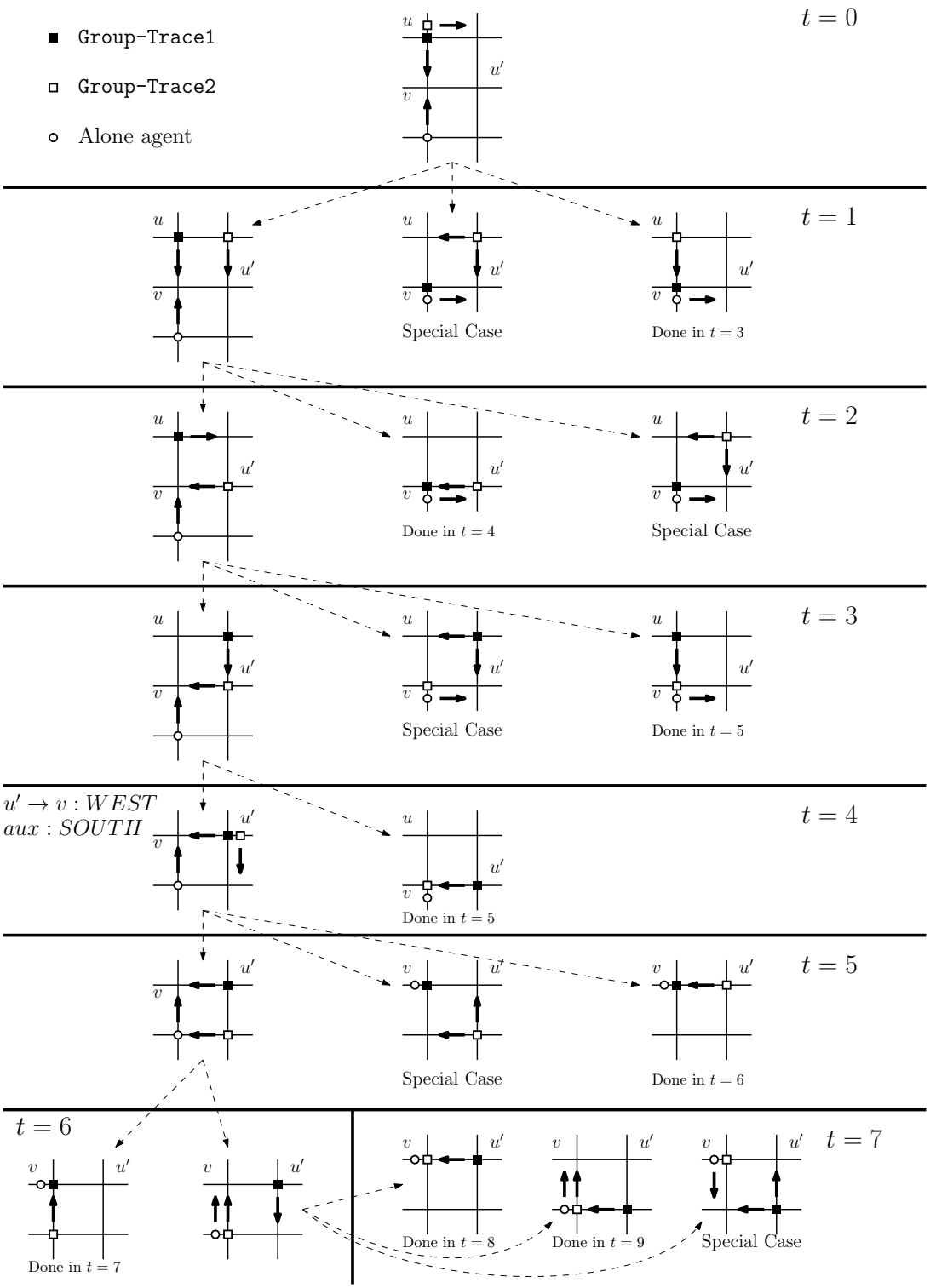

FIGURE 3.19: This figure depicts the algorithm of two agents in states `Group--Trace1` and `Group-Trace2` respectively. In this figure the group of agents is moving towards the south. The case where the group moves towards the north and the alone agent moves towards the south is analogous. Notice that at $t = 4$ `Group-Trace1` and `Group-Trace2` start the algorithm from the start, but this time their intended direction is West and their auxiliary line is to the South.

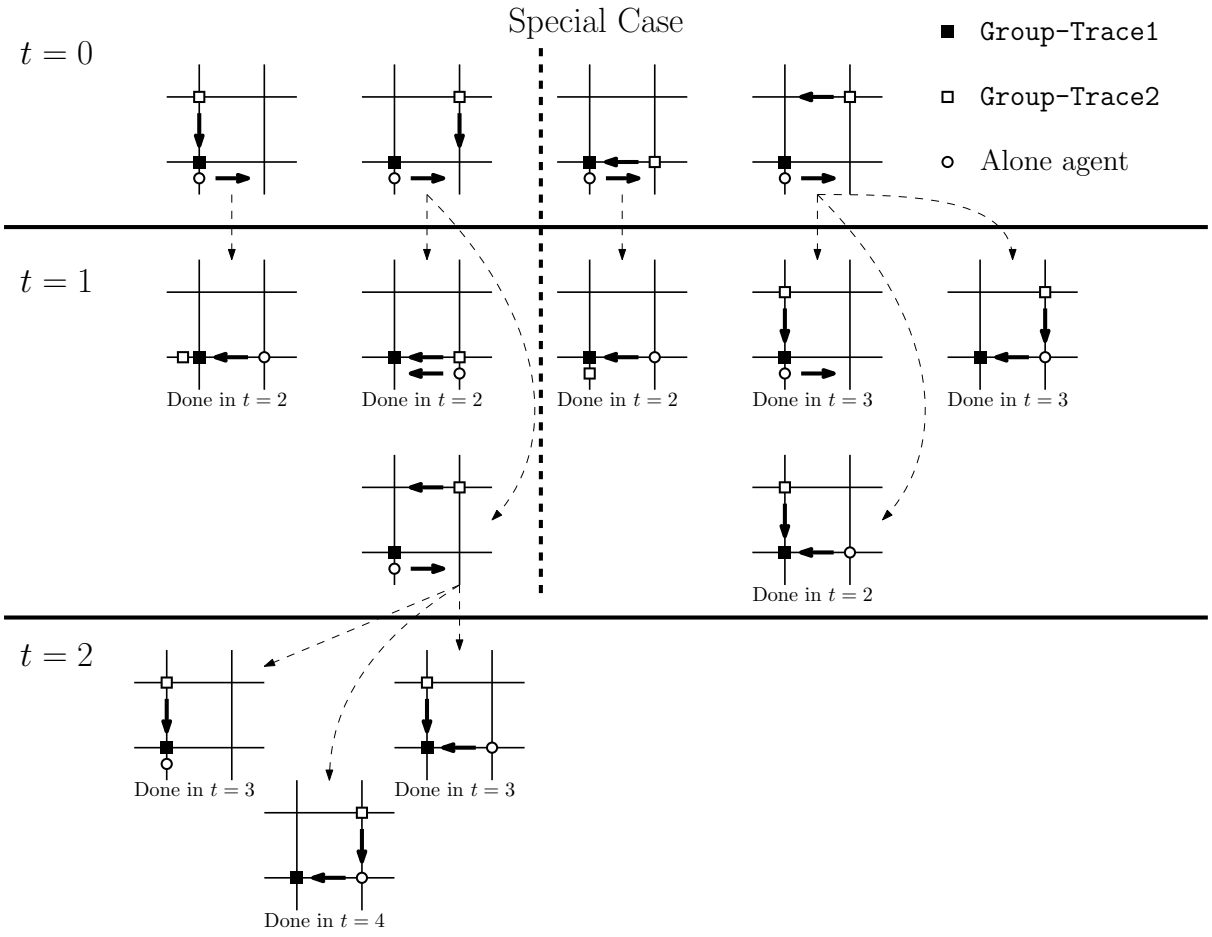

FIGURE 3.20: This figure depicts the algorithm for the special case of Fig. 3.19.

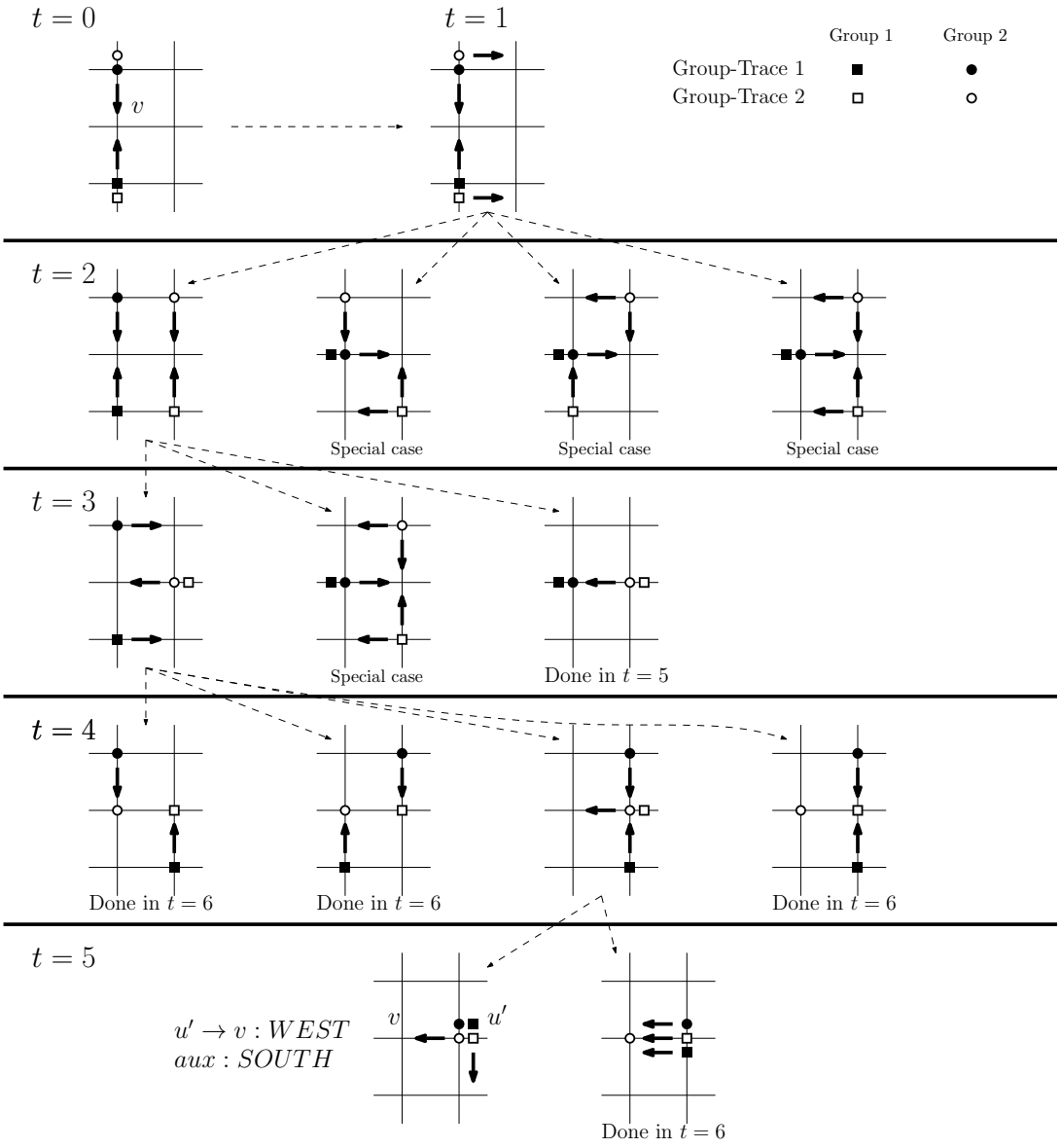

FIGURE 3.21: In this figure we can see two groups trying to move to the same node. One is moving South and the other is moving North.

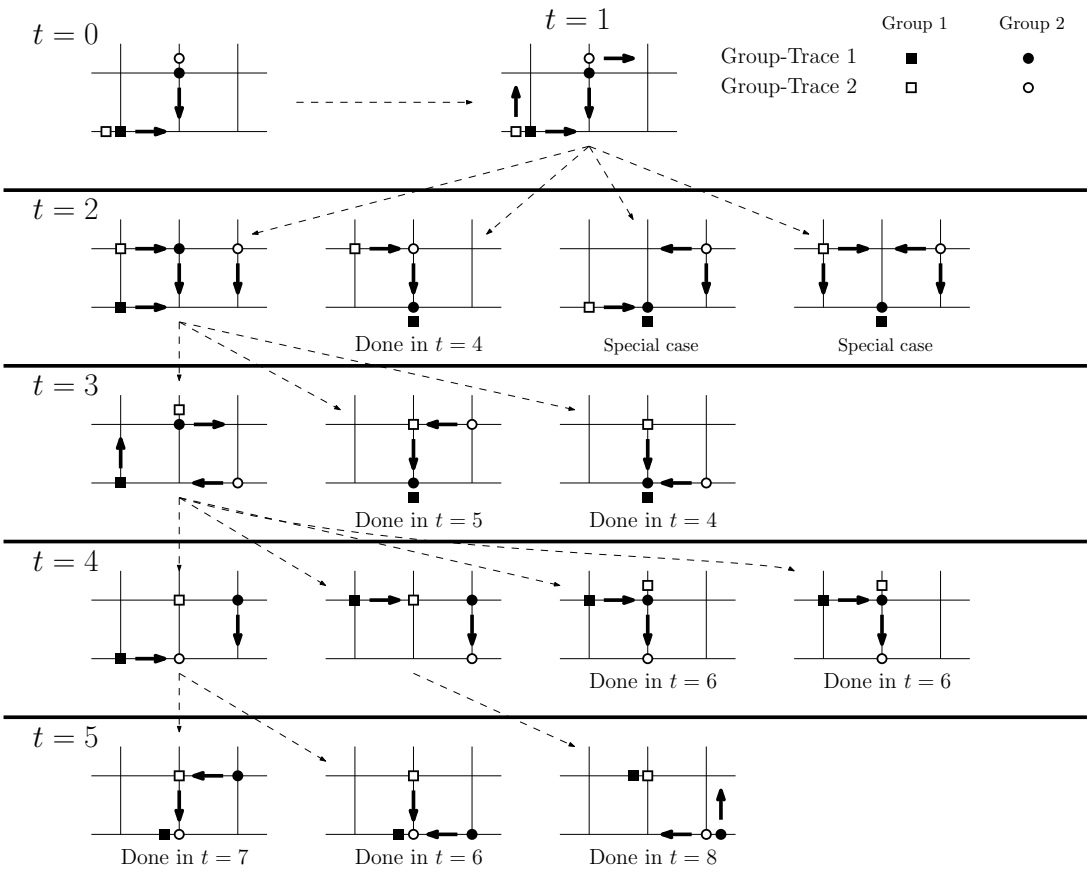

FIGURE 3.22: In this figure we can see two groups moving to the same node. One is moving South and the other, changing column, is moving East.

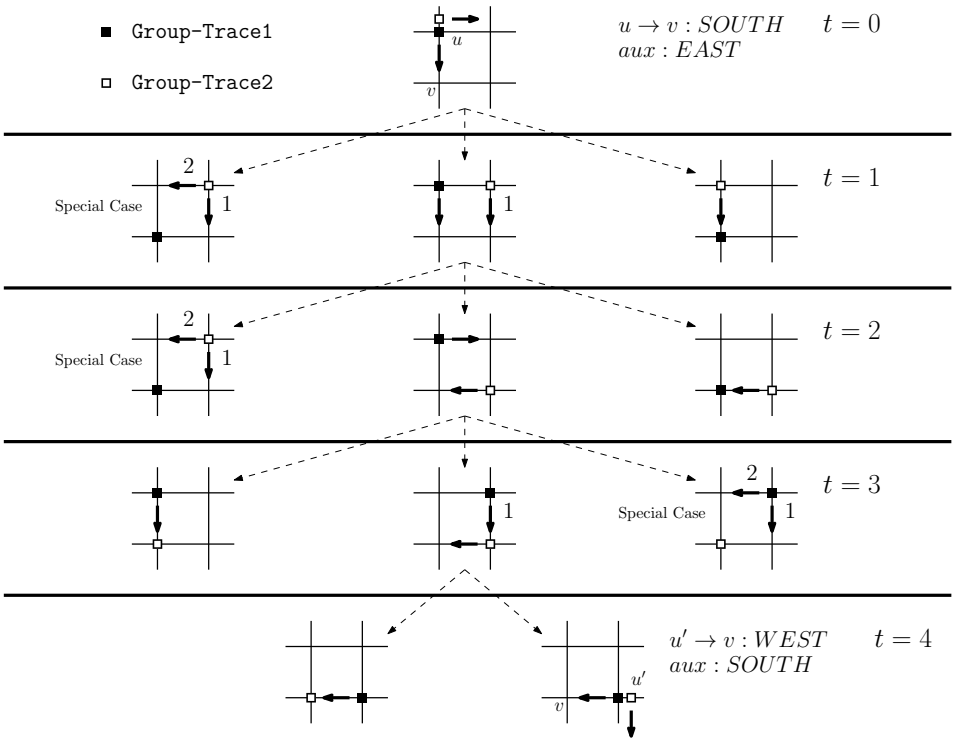

FIGURE 3.23: Two agents which consist a group of explorers are blocked at a routing distance more than two from the alone agent. **Note:** In the cases depicted as *Special Case* in this figure, the group agent associated with two outgoing arrows repeatedly tries directions 1 and 2 until it succeeds. Since there is at least one other agent besides this group of agents, either this third agent or the group agent will move. Finally the third agent will either meet the group agent on the bottom left node (see Figure 3.20) or the group agent on the upper right node. On the latter case, the agents choose distinct identities and move in different directions in order two create a path towards the bottom left group agent.

3.3 Discussion

Comparing our results with the previously known results for asynchronous agents in an oriented grid, we notice that $k \geq 3$ synchronous agents with only local visibility capability can gather starting from any configuration without multiplicities, while for asynchronous agents it has been previously proved that the agents can only gather if they can "see" at distance two and they start from a connected configuration. On the negative side we proved that two synchronous agents cannot gather even if they have global visibility capability.

When the synchronous agents start from a configuration with multiplicities (i.e., including nodes with more than one agents), then the problem is unsolvable in the general case, since the initial configuration could be such that the agents form two groups located at two distinct nodes and they behave like two agents.

Some interesting directions include the study of the problem in other network topologies such as oriented multidimensional grids, well structured graphs that are easy to explore by constant memory agents, and arbitrary graphs where agents have a map. Another scenario is one with a less powerful malicious agent which has limited speed capabilities. It would be interesting to see whether the impossibility results could be circumvented in this case, and to investigate whether the problem could be solved even in the presence of multiple malicious agents.

4 Black Virus Decontamination

In this chapter we discuss the black virus decontamination problem. We often need to solve problems in networks when hostile entities are present which may harm the system. A model of a hostile static entity which is called *black hole* has been defined and extensively studied. A black hole is assumed to be a node which is infected by a malicious process that destroys any incoming agent without leaving any trace, and the goal is for a group of agents to locate the black hole within finite time. Another type of a malicious mobile entity is the *intruder*. In the *intruder capture* problem (also known as *graph decontamination* and *connected graph search*) a harmful agent, called the intruder, can move in the network and infect the visited nodes. The objective is for a team of mobile agents, that cannot be harmed by the intruder, to decontaminate the network. Hence a black hole is harmful to the mobile agents but not to other nodes of the network, while an intruder damages nodes but not agents.

We study here a model of another hostile entity called *Black Virus* which has been introduced by J., Cai et al in 2014. In particular we study the *Black Virus Decontamination* (BVD) problem within the distributed computing and especially the mobile agents' area. The Black Virus is a malicious entity similar to a *black hole* which is initially located at a node u of the network and has the following behaviour: when a mobile agent enters node u, the agent is removed from the network without leaving any trace. The Black Virus spreads to all unoccupied by agents neighbouring nodes of u, effectively expanding its contamination over the network, but node u is now clean (decontaminated). The goal is to find the minimum number of agents that can decontaminate a given network with a Black Virus initially located at an unknown place and design a fast distributed algorithm for a certain (preferably weak) model of mobile agents.

Hence the black virus model combines some of the characteristics of both the black hole

and the intruder. More specifically, the black virus combines the threat of a harmful node with the need for decontamination and the ability to infect additional nodes.

In order to prevent the Black Virus from spreading the agents must occupy the node(s) where the Black Virus is going to spread to. Hence the only way to clear a contaminated node and at the same time prevent it from spreading the contamination, is to have the node visited by an agent while at the same time all adjacent nodes are occupied by agents.

We study the black virus decontamination problem on synchronous ring graphs. More specifically, a ring (or cyclic defined in 2.1) graph is an undirected simple graph $G = (V, E)$ where each vertex $u \in V$ has exactly two neighbours. The nodes of the ring are anonymous. The ring initially contains a number of mobile agents, initially located at distinct nodes, and one black virus. All edges incident to a node have distinct port labels, visible to an agent at the node. In the case of an oriented ring the port labels at each node are globally consistent and all agents agree in a common sense of direction. In the case of an unoriented ring the port labels at each node are locally consistent, hence when two agents which are initially located at different nodes choose the same port label, they might choose different directions. Thus, in this case it might happen that not all agents perceive the same direction as clockwise (resp. counter-clockwise) direction.

The ring is synchronous meaning that an agent needs one unit of time to traverse an edge. The time an agent needs in order to compute its next move is negligible.

The mobile agents are computational entities that operate in the network and are able to move from one node to another. The agents can only move from a node to a neighbouring one, in one time unit. The agents may have distinct identities. Most of our negative results hold even for agents with distinct identities. Our algorithm for oriented rings, although it is initially presented (for convenience) assuming that the agents have distinct identities, as we later describe, it can be slightly modified to work for anonymous agents.

The agents are identical (apart from their identities when they are distinct) and they are equipped with movable identical tokens, which they can leave at (or pick-up from) nodes. In all our algorithms each agent has at most one token. An agent can communicate with other co-located agents. More specifically, when two (or more) agents are at the same node they can read each other's state, as in the face-to-face model of communication

described in Chapter 2. Moreover, they can read each other's identity (only in the case of having distinct identities). They cannot exchange messages. Their memory, although it is enough for reading other agents' identities when needed, it is not related to the number of nodes of the ring and therefore for example the agents cannot count more than a constant number of nodes. The agents can only meet at nodes and not inside edges, i.e., two agents traversing the same edge do not notice each other.

All the agents are initially scattered in the network and they execute the same deterministic protocol starting at the same time. The agents do not initially have any knowledge about the size of the ring or their configuration.

The black virus is a malicious entity which is initially placed at an unknown location in the ring. The interaction between the agents and the Black Virus is the following. When an agent decides at time t to move to a node w where the Black Virus resides three events occur. At time $t+1$ the agent vanishes without a trace, node w is cleaned, and the virus copies itself to any neighbouring nodes of w which are unoccupied by agents at time $t+1$. Notice that if an agent tries to move to a node v and at the same time a clone of the black virus tries to move to v, then the agent moves to v while the clone does not. If a clone of the Black Virus moves to a node (unoccupied by agents) where a clone already exists (or moves there at the same time), then the two clones merge to one. In other words any node can contain at most one clone of the Black Virus.

The goal of the Black Virus Decontamination problem is to design an algorithm which eliminates any clones from the network within finite time and at least one agent survives. The agents are able to coordinate using the $\mathcal{T}$OKEN model defined in 2.2.1. The agents may have one or more tokens which can be put down or picked up by any agent in the ring.

Preliminary results of this chapter have appeared in [56] and more results can also be found in [62].

4.1 Decontaminating an oriented ring

4.1.1 Impossibility results

The following simple observations easily lead to a few impossibility results.

Consider a ring with a black-virus located at a node u. The only way to clean the black-virus, demands having an agent visiting the contaminated node u. It is obvious that one agent is not enough to decontaminate the ring, even if the nodes of the ring are distinctly labelled (in that case we say that the ring is labelled) and the agent knows exactly the location of the black-virus, since the agent will vanish and the virus will spread to the adjacent nodes of u. Two agents are also not enough to solve the problem, even if the ring is labelled and the agents know exactly the location of the black-virus, since at least one agent will vanish and at that time at least one of the adjacent nodes of u will be unoccupied by agents. Hence the virus will spread there and due to the previous observation, the (only one) remaining agent cannot decontaminate the network.

Lemma 4.1. *Consider a synchronous, oriented and labelled ring whose one node is contaminated by a black-virus. Then 2 agents are not enough to decontaminate the ring, even if they have distinct ids, an unlimited number of tokens, unlimited memory and know the initial configuration on the ring (i.e., they have an exact map of the ring and know the initial locations of all agents), and the location of the black-virus.*

Consider an interval of $x \geq 2$ consecutive nodes (i.e., an interval of length $x-1 \geq 1$ edges) in a synchronous, oriented and labelled ring so that its two endpoints u, v are occupied by clones of the virus and no node of the interval is occupied by an agent. We call such an interval a *contaminated interval* (notice that any other node of the interval apart from its endpoints could be either clean or not). Then the only strategy which can be used to decrease the length of the contaminated interval, is first having an agent located at a node w which does not belong in the interval and is adjacent to an endpoint, say u. We call it *guarding* action. Then another agent should visit node u (we call it *attacking* action), and hence u will be cleared and since w is guarded, the length of the contaminated interval will be decreased. The *guarding* action should take place either before or at the same time as the *attacking* action. We will later define formally a combination of those two actions in order to effectively scan unexplored nodes of the network.

Lemma 4.2. *Consider an interval of $x \geq 2$ consecutive nodes in a synchronous, oriented and labelled ring so that the two endpoints of the interval are occupied by clones of the virus and no node of the interval is occupied by an agent. Then $x+1$ agents are not enough to decontaminate the interval, even if they have distinct ids, an unlimited number of tokens,*

unlimited memory and know the initial configuration on the ring (i.e., they have an exact map of the ring and know the initial locations of all agents) and the exact location of the contaminated interval.

Proof. While the contaminated interval consists of $x \geq 2$ nodes, the only strategy that can be used in order to decrease its length l ($l = x - 1 \geq 1$) should be a combination of an attacking and a guarding action as described before. Hence at least one agent should vanish in order to decrease the length of the interval by one node. Finally, when the interval has been decreased to just one contaminated node, in view of Lemma 4.1, the remaining two agents are not enough to completely clean the black-virus. □

An immediate consequence of Lemma 4.2 is the following lemma which gives us a better lower bound on the number of agents when the location of the black-virus is not initially known, even for a powerful model of agents.

Lemma 4.3. *Consider a synchronous, oriented and labelled ring which is contaminated by a black-virus. Then five initially scattered agents are not enough to decontaminate the ring, even if they have distinct ids, an unlimited number of tokens, unlimited memory and know the initial configuration, i.e., have an exact map of the ring and know the initial locations of all agents.*

Proof. Suppose for the sake of contradiction that there is an algorithm that solves the problem in any ring and for any initial configuration (and initially unknown location of a black-virus). Consider an initial configuration in which the distance d^0_{ij} between any two agents i, j at time $t = 0$ is greater than 3 edges. The algorithm should eventually move at least one agent. Let A be the first agent that moves from its initial location u to an adjacent node v at time t (if more than one agents simultaneously move then let A be anyone of them). Then the adversary places the virus at v and agent A vanishes. Since the distance $d^t_{Aj} \geq d^0_{Aj} - 2 \geq 2$, where j is any of the two agents closest to A, the adjacent nodes u, w of v cannot be occupied by agents other than A at time t or before. Hence clones of the virus contaminate the nodes u, w creating a contaminated interval which consists of 3 nodes: u, v, w as shown in Figure 4.1. In view of Lemma 4.2, the four remaining agents are not enough to decontaminate this 3-node interval. Hence five agents are not enough to decontaminate the ring. □

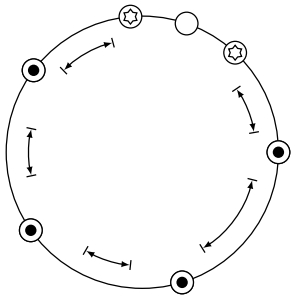

FIGURE 4.1: Five agents are initially scattered on a ring containing a virus. The first agent that moves, vanishes and the resulting configuration is shown here.

Moving to weaker agent models and following a similar reasoning we can easily improve the lower bound from 5 to 7 when the agents do not know their initial configuration.

Lemma 4.4. *Consider a synchronous, oriented and labelled ring which is contaminated by a black-virus. Then seven initially scattered agents are not enough to decontaminate the ring, even if they have distinct ids, an unlimited number of tokens, unlimited memory and know the size of the ring.*

Proof. Suppose for the sake of contradiction that there is an algorithm that solves the problem in any ring and for any initial configuration and consider an initial configuration in which the distance between any two agents is greater than 3 edges. The algorithm should eventually move at least one agent A. Similarly as in the proof of Lemma 4.3, after the first move of the first agent A that moves, this agent vanishes along with its tokens (even if some of them had been left at its initial position) and the configuration is like the one shown in Figure 4.2. Suppose the initial configuration was selected so that for the distances x, y in Figure 4.2 it holds $x \geq 2y$ and $y \geq 1$.

The algorithm should eventually move at least one of the remaining agents for at least y nodes on the same direction in order to meet with another agent, meet another agent's token or approach the contaminated area. The agent B that first moves such a distance can always be selected by the adversary to be one of the agents closest to the contaminated interval in such a way that this agent vanishes (possibly leaving many tokens), and the contaminated interval now consists of at least 4 nodes, since the closest agent to B towards the safe area is at a distance greater or equal to $x \geq 2y \geq 2$ (see Figure 4.3).

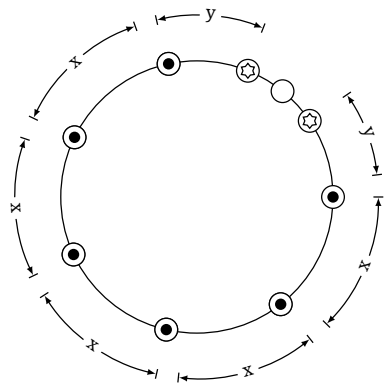

FIGURE 4.2: Seven agents are initially scattered on a ring containing a virus. The first agent that moves, vanishes and the resulting configuration is shown here.

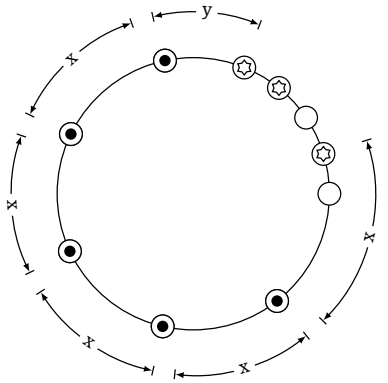

FIGURE 4.3: Seven agents are initially scattered on a ring containing a virus. The resulting configuration after two agents have met the black-virus.

In view of Lemma 4.2, the remaining 5 agents cannot decontaminate the interval consisting of 4 nodes and therefore the algorithm cannot decontaminate the network. □

In order to prove the impossibility result for 7 agents (Lemma 4.4) it was enough to initially place the agents so that at least two of them vanish while the black-virus spreads to a contaminated interval consisting of at least four nodes. Then Lemma 4.2 applies. If however more than 7 agents are initially available, then after 2 agents vanish, the contaminated interval consisting of at least four nodes combined with the fact that the number of the remaining agents in that case is more than 5 do not immediately permit the application of Lemma 4.2. Moreover, some tokens (which belonged to agents that vanished) might have been left over, and could potentially help the remaining agents to decontaminate the ring. Nevertheless with a more careful initial placement of the agents, we can show that for any possible algorithm there is an initial placement of nine agents so that, the first two agents vanish leaving behind at most one token which cannot

help the remaining agents to form a group of at least two agents before a third agent also perishes thus increasing the length of the contaminated interval to 5 nodes. Hence, due again to Lemma 4.2, 6 remaining agents cannot solve the problem.

Lemma 4.5. *Nine agents are not enough to decontaminate an $n-$node, synchronous, oriented and labelled ring, even if they have distinct ids, an unlimited number of tokens, unlimited memory and know the size of the ring.*

Proof. Suppose for the sake of contradiction that there is an algorithm $\mathcal{A}$ that solves the problem in any ring and for any initial configuration. Consider the first two agents A, B which are instructed to move by Algorithm $\mathcal{A}$ (if more than two move at the same time take any two of them). Agents A, B should either simultaneously move, or one of them first moves and then, within a finite time, a second one moves. Notice that an algorithm that does not move a second agent within a finite time, cannot be a correct algorithm, since the adversary can initially place the agents so that the first one immediately vanishes and then nobody moves.

First consider the case where the agents A, B do not move simultaneously. Let A be the first agent that moves from node u to an adjacent node v. The adversary initially places the black virus at v and agent A vanishes along with its tokens (even if some of them were placed at u before the agent moves). Now the contaminated interval extends to the two neighbouring nodes u, w of v. Agent B moves within a finite time. According to the direction agent B selects, the adversary can select its initial position to be at an adjacent node of u or w different than v, and therefore B vanishes after its move at time t along with its tokens (even if some of them were placed at its initial position). The contaminated interval at time t has increased to 4 nodes. A third agent should be instructed to move either simultaneously with agent B or within a finite time after agent B's move. If the remaining agents have been initially placed so that the distance between any two of them is more than 3 edges, and two of them C, D occupy nodes at a distance of at most two edges at time t from the endpoints of the current contaminated interval, then the first agent that moves a distance of at least 2 edges towards one direction (there must be such an agent, otherwise the agents will never meet other agents or tokens different than their tokens, or approach the contaminated interval) is selected to be one of the agents C

or D and therefore vanishes. The contaminated interval increases to 5 nodes, while the remaining agents are 6. Hence due to Lemma 4.2, the 6 remaining agents cannot solve the problem.

Now consider the remaining case where the agents A, B move simultaneously. If those are the only agents that move at the same time and they move towards different directions then the adversary can initially place them at a distance two at nodes u, w and place the black virus at the common neighbour v of u, w. Therefore both agents along with their tokens (even if some of them were placed at nodes u, w) are vanished. Now the remaining 7 agents are in their initial configuration and they need to clean a contaminated interval of 3 nodes which is impossible since they cannot solve the problem even when there is only one contaminated node as we showed in Lemma 4.4. In the remaining subcases, either agents A, B are not the only ones that moved simultaneously, or they moved towards the same direction. Notice that if more than two agents moved simultaneously then at least two of them moved towards the same direction. Hence in all the remaining subcases there are at least two agents that moved simultaneously towards the same direction. Let A, B be two agents that moved simultaneously towards the same direction. If among all agents that moved simultaneously there is at most one agent A that left some of its tokens before its move, then the adversary can initially place the agents so that agent A vanishes along with its tokens (even if some of them were left at its initial location) and two other agents occupy the two nodes adjacent to the endpoints of the contaminated interval which now consists of 3 nodes. Hence, the next agent which moves can be selected by the adversary to be one of those two agents, which vanishes along with its tokens. Now there are 7 remaining agents with a contaminated interval consisting of 4 nodes (see Figure 4.4). The configuration is similar to the one in Figure 4.2, but with 7 agents and a contaminated interval consisting of 4 nodes. Following the same arguments as in the proof of Lemma 4.4, we can show that the agents cannot solve the problem. Finally, suppose that there are at least two agents A, B that leave some of their tokens and move simultaneously towards the same direction (say counter-clockwise without loss of generality). The adversary can arrange the initial positions of the agents so that agent A vanishes along with its tokens, while the tokens that

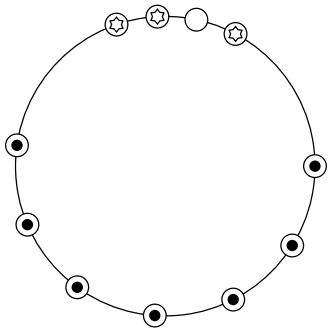

FIGURE 4.4: Nine agents are initially scattered on a ring containing a virus. This is the resulting configuration after two agents have met the black-virus without leaving any tokens behind.

agent B had left, are vanished and B is now located next to a contaminated node. A resulting configuration is shown in Figure 4.5, where Δ denotes the nodes where the agents were initially located. If the next agent moves clockwise then the adversary can select

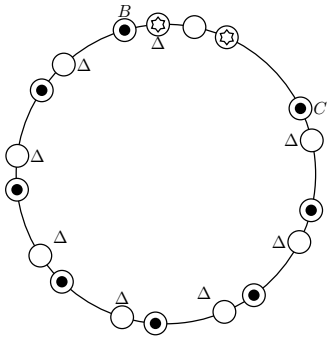

FIGURE 4.5: Nine agents are initially scattered on a ring containing a virus. This is the resulting configuration after one agent has met the black-virus. The nodes denoted with Δ clockwise next to each agent are the initial homebases of the respective agents. The initial homebase of agent B is now contaminated and therefore any token that had been left there by B has disappeared.

B as the next agent and the situation is exactly like in the previous case with 7 agents and a contaminated interval consisting of 4 nodes and can be again treated in the same way. If the next agent moves counter-clockwise then the adversary can select C as the next agent (which was initially located at a distance 2 clockwise from agent A) which vanishes and the contaminated interval consists of 4 nodes. Now each of the remaining agents can safely move clockwise until one node before its respective initial homebase (Δ) (without reaching it) and counter-clockwise until the closer (another agent's) initial homebase. Notice that all those intervals do not have a node in common. Therefore at least one agent should eventually move to a node outside its safe area (otherwise no agent can meet any other agent or the black-virus). The first agent that moves clockwise

(respectively counter-clockwise) to a node outside its safe area is selected to be agent B (respectively the next agent clockwise of C) and vanishes. Due to Lemma 4.2 the remaining 6 agents cannot clean a contaminated interval consisting of 5 nodes. □

4.1.2 An algorithm with ten agents

In this section we present an algorithm that decontaminates an oriented ring using ten agents. In order to make the presentation easier, we first describe the algorithm for agents with distinct identities and then we discuss how this algorithm can work with anonymous agents.

Before describing the algorithm, we define the `Cautious-Move` procedure, which we use in all our algorithms. This procedure is a combination of an attacking and guarding action which is used by a group of at least two co-located agents in order to scan unexplored nodes and decrease the length of a contaminated interval, as follows. One of the agents (leader) of the group located at a node u at time t, moves to an adjacent node v while the remaining agents of the group (companion) wait at u. Hence at time $t+1$ the leader agent is at v while the remaining agents of the group are at u. In the next time unit all remaining agents of the group move to v while the leader (if it is still alive) waits at v.

Procedure Cautious-Move(dir)

```
1 if leader then
2     Move 1 step dir
3     Wait(1)
4 else if companion then
5     Wait(1)
6     Move 1 step dir
```

It is easy to see that if node v was safe when the leader moved there then all agents of the group gather at v within two time units. If there was a black-virus at v when the leader moved there, then the leader has vanished but the virus did not spread to node u, and node v is now safe. Hence if the companion agents do not meet the leader at node v, they can conclude that the virus was at v. Furthermore, if v was an endpoint of a contaminated interval consisting of at least two nodes, now its length has been decreased by one.

The general idea of the algorithm which cleans any oriented ring is the following. First each agent tries to discover whether there is another agent at distance one. Then each

agent leaves its token and moves clockwise until it finds a token. This is a special case which we handle separately in the algorithm. Now each agent moves back to its home-base, picks up any tokens, and moves clockwise again until it finds a token. This time it waits at the node with the token until another agent comes. Now the group scans the ring, using Procedure `Cautious-Move`, until it meets the black-virus. Then the remaining agent of the group waits there forever. The rest of the agents repeatedly form groups of at least two agents and scan the ring in order to find the other endpoint (i.e., other than that guarded by an agent) of the contaminated interval and decrease one-by-one its length. The algorithm takes care of situations where some agents were initially located at distance one, or an agent which belongs to a group meets another agent, etc.

Procedure Initialize

```
1  Release token
2  Move one step cw
3  if there is a token on the node then
4  |  pick up token
5  |  move one step ccw
6  |  release token
7  |  wait(1)
8  |  if there is another agent on the node then state ← leader
9  |  else state ← explorer
10 else
11 |  move one step ccw
12 |  if no token on the node then
13 |  |  move one step ccw
14 |  |  if no token on the node then  state ← guard
15 |  |  else  state ← companion
16 |  else
17 |  |  state ← explorer
18 return state
```

Lemma 4.6. *At most three agents are lost until at least two agents meet. The length of the contaminated interval can be increased to at most five nodes.*

Proof. We will first show that at most two agents can be lost while executing Procedure `Initialize`, increasing the length of the contaminated interval to at most 4 nodes. Then we will show that at most one more agent can be lost while executing Procedure `Explorer`, increasing the length of the contaminated interval to at most 5 nodes. First, all agents begin executing Procedure `Initialize`. During `Initialize` an agent first moves

Procedure Explorer

```
1  Move cw until you find a token and return to your homebase
2  Pick up token(s)
3  if there is another agent on the node then execute Leader(cw)
4  repeat
5      Move 1 step cw
6      v ← current_node
7  until (a member of a pair moving ccw is encountered) OR (a token is detected)
8  if there is a token at v then
9      if there is a companion at v then
10         execute Leader(cw)
11     else
12         Switch to state companion and wait until an agent gets to v
13         execute Companion(dir_of_leader)
14 else
15     execute Companion(ccw)
```

Procedure Companion(dir)

```
1  state ← companion
2  while true do
3      execute Cautious-Move(dir)
4      if dir = cw then
5          if leader not on node then execute Guard()
6          else if guard on node then dir = ccw
7      else if dir = ccw then
8          if leader not on node then
9              Companion with smallest ID becomes leader
10             if you became the leader then execute Leader(ccw)
```

one step clockwise and then one step counterclockwise. During those two steps at most two agents might vanish. This case may only occur when one agent vanishes during the first step and a second agent vanishes during the second step. Notice that some agents may perform an additional counterclockwise step. However, no agent can be lost during this step. Let us suppose that there is an agent A_1 that performs the additional counterclockwise step. This means that A_1 did not encounter a token during the clockwise step and another agent, A_2, that is at distance one from A_1, moved the token left by A_1. If A_2 is also the agent that is lost during the second step, A_1 moves to a node incident to the contaminated interval and becomes a guard. If A_2 is not one of the agents that are lost then A_1 moves to the same node as A_2 and the two agents form a pair. In either case, A_1 is not lost. Next, the agents begin executing `Leader, Companion, Guard` or `Explorer`. The

Procedure Leader(dir)

```
1 state ← leader
2 Let v be the current node
3 while true do
4 |   execute Cautious-Move(dir)
5 |   if dir = cw then
6 |   |   if you meet another leader on v then execute Companion(ccw)
7 |   |   else if guard on v then dir = ccw
```

Procedure Guard

```
1 state ← guard
2 Let v be the current node
3 repeat
4 |   Wait on v
5 until leader at v AND dir_of_leader = ccw
6 execute Companion(ccw)
```

agents executing `Leader` or `Companion` always move using `Cautious-Move` and therefore even if an agent is lost, the contaminated interval is decreased. Furthermore, an agent executing `Guard` waits on a safe node and therefore is not lost. Let us now consider the agents executing Procedure `Explorer`. Those agents move clockwise until they detect a token. Exactly one agent that is moving towards the black virus in the path that does not contain a token is lost. Notice that the token that was left by the agent that was lost cannot be destroyed, therefore all other agents will reach a token and eventually at least two agents will meet.

The first agent that is lost increases the size of the contaminated interval from one to three nodes[1]. Each additional agent that is lost infects one node that is already in the contaminated interval and one previously clean node. Therefore, each agent that is lost increases the size of the contaminated interval by one node. After at most three agents are lost, the contaminated interval consists of five nodes. Finally, any other agent that visits an infected node is executing Procedure `Cautious-Move` and will decrease the size of the contaminated interval. □

[1] Recall that in a contaminated interval, clean nodes might be enclosed between infected ones. However, those clean nodes can be considered infected as well since they will get infected while the agents try to decontaminate the interval.

Algorithm 5: ScatteredBVD

```
1 State ← Initialize
2 switch State do
3     case leader do Leader(cw)
4     case companion do Companion(cw)
5     case explorer do Explorer
6     case guard do Guard
```

Lemma 4.7. *The algorithm* `Scattered BVD` *solves the black virus decontamination problem in any synchronous oriented ring consisting of* n *nodes within* $O(n)$ *time units, using ten scattered agents with constant memory and one token each.*

Proof. First the agents begin executing Procedure `Initialize`. As we showed in Lemma 4.6, at most two agents can be lost during this step. The agents that did not form a pair during Procedure `Initialize`, begin executing Procedure `Explorer`. During this procedure at most one agent can be lost as we showed in Lemma 4.6. Each one of the remaining agents executing Procedure `Explorer` reaches a token while moving clockwise, return to its homebase, picks-up its own token and once again moves clockwise until it reaches a token. The agents that reach a token for the second time are at a node containing a token left either by an agent that has not yet returned or by the agent that was lost. In the first case, another agent will eventually return and the two agents will form a pair. In the second case, since the agent that is waiting has removed its own token, at least one more agent will reach the same node. Therefore, eventually at least one pair is formed. The agents executing Procedures `Leader` or `Companion` move using Procedure `Cautious-Move`. As we showed in Lemma 4.6, the contaminated interval consists of at most five nodes. At some time t, a pair will reach the contaminated interval for the first time. The leader of the pair will be lost, the contaminated interval will be decreased by one node and the companion will become a guard when it detects the loss of the leader. Any other pair moving clockwise will reach the guard and change direction, therefore no additional agent moving clockwise will be lost. Furthermore, any agent that encounters a pair or a team of agents moving counterclockwise begins executing `Companion` and follows the leader of the team. Therefore, eventually all the remaining agents in the ring except the guard (i.e., six agents), will be part of the same team, moving counterclockwise. At most

four of those agents will be lost decreasing the length of the contaminated interval (consisting of at most five nodes) to one infected node. Let v, u and w be three consecutive nodes and let u be the last infected node in the ring. The *guard* is located at v and the last *leader* and *companion* are located at w. In the last step of the algorithm, the leader moves to the infected node and is destroyed by the black virus. The black virus copies itself and attempts to spread to v and w. However, both nodes are already occupied by the last two agents and the ring is decontaminated. It is easy to see that the agents travel a distance equal to at most a constant times the length of the ring. □

Although, the algorithm above has been described for agents with distinct identities, it is easy to slightly modify it so that it works for anonymous agents. Notice that the agents use the distinct identities in order to assign different roles to themselves only when they are co-located. Hence, a mechanism could be used to help co-located agents to assign themselves different roles. For example, as soon as (i.e., the first time that) two agents occupy the same node u and they are at the same state, they may differentiate themselves according to the direction by which they entered node u and the actions they were doing one time unit before. Notice that if the agents entered u through the same edge, they have the same state and they were not co-located before, then exactly one of them was moving to u one time unit before, while the other one was already at u.

4.2 Decontaminating an unoriented ring

4.2.1 Impossibility results

Suppose that the agents do not agree on the clockwise direction of the ring. Naturally, all impossibility results for oriented rings hold for unoriented rings. We first show that ten agents with distinct identities and an unlimited number of tokens cannot solve the problem. We then show that eleven agents with distinct identities and one token each are also not enough. We also show that if the agents are anonymous then they cannot solve the problem no matter how many they are and how many tokens they have.

Lemma 4.8. *Ten agents are not enough to decontaminate any synchronous, unoriented ring, even if they have distinct identities, an unlimited number of tokens and unlimited memory.*

Proof. Let A and B be two agents that are located at the two nodes which are adjacent to the one containing the black virus. Furthermore, let C and D be two agents that have been placed so that the initial distance between A and C and between B and D is two edges. Finally, let no other agent be located between A and C or B and D in the part of the ring that does not contain the black virus. We will first consider the case in which at least two agents begin moving simultaneously. The adversary can pick the direction in which the agents move so that both A and B will begin moving towards the infected node. A and B are destroyed in the first step along with their tokens, and the contaminated interval consists of three nodes. Notice, that their tokens are destroyed regardless of whether they were placed before A and B moved. Now, C and D are located closest to the contaminated interval. The adversary can always pick C and D as the next agents to move a distance of at least two edges in the same direction so that both agents are lost, while their tokens possibly survive. The length of the contaminated interval has been increased to five nodes. The six remaining agents cannot clean a contaminated interval of five nodes as we showed in Lemma 4.2.

If only one agent moves first then there are nine remaining agents (at their initial positions) with a contaminated interval consisting of three nodes. Hence in view of Lemma 4.5 the problem is unsolvable. □

Lemma 4.9. *Eleven agents with distinct IDs and one token each do not suffice to decontaminate an unoriented synchronous ring.*

Proof. If at most two agents first move simultaneously then the adversary initially places the agents so that they meet the black-virus and vanish after their first step. Hence at most ten agents remain located in their initial nodes while the contaminated interval consists of three nodes. Thus in view of Lemma 4.8 the problem is unsolvable.

Suppose now that at least three agents first start to move simultaneously and consider the following initial configuration. Two agents, A and B are initially placed at nodes

which are adjacent to the node containing the black virus. Another agent C is the closest neighbour of A in the part of the ring that does not contain the black virus, and agent D is the closest neighbour of B in the part of the ring that does not contain the black virus. The adversary arranges chirality of agents so that agent A will move towards the black virus, agents B and C will move in the same direction as A and D will move in the opposite direction. The adversary selects all those four agents (if at least four move simultaneously) or A, B and C (if only three first move) to move simultaneously. The agents might place their token, and begin moving. Notice that as long as the agents do not place their tokens on a node, when an agent is lost, its token is lost as well and the surviving agents do not learn anything about the location of the black virus or of the other agents. Therefore we can assume that the agents place their token, and begin moving. After the first step, A along with its token and the token left by B are destroyed and the black virus copies itself and infects its neighbouring nodes. The remaining agents can either move for a specific number of steps or move until they detect a token or an agent. Notice that if the agents do not move, the ring cannot be decontaminated. If the agents only move for a specific number of nodes and return to their respective homebase, the ring will not be decontaminated. Therefore, the agents need to move until they find a token or an agent. If the agents move then agents C and D are destroyed, the black virus infects two additional nodes and B reaches the token left by D. Now, there are five nodes that need to be cleaned, eight agents and nine tokens in the network. Agent B can either move past the token left by D, remain at the node containing the token or change direction and continue moving. If B stops moving when it encounters a token, all the agents in the ring will eventually reach a token and stop moving. If B continues moving past the token, B will not be harmed by the black virus, but some other agent will also move past the tokens left by D and C and be destroyed by the black virus. Hence, there will be seven remaining agents and in view of Lemma 4.2 the problem is unsolvable in this case. Let us consider the case in which B changes direction and continues moving. If B only moves for a number of steps less than the distance to its homebase, all other agents will also move likewise and the agents will not be able to meet, since no two agents will visit a common node in this case. Therefore, B needs to move until it reaches its homebase. However, B will be destroyed by a black virus before reaching its homebase. Now, there are seven agents and a contaminated interval consisting of six nodes in the ring. The

remaining agents are not enough to decontaminate the ring as we showed in Lemma 4.2. □

Lemma 4.10. *An unoriented synchronous ring cannot be cleaned by any number of anonymous agents even if they have an unlimited number of tokens and unlimited memory.*

Proof. Assume an unoriented ring with k anonymous agents and consider the starting configuration of Figure 4.6 (if k is even) or of Figure 4.7 (if k is odd). If the initial number of agents is even, the ring consists of $n = k + 1$ nodes and there are initially k agents and one black virus. Each node is initially occupied by one agent, except the node that contains the black virus (see Figure 4.6). If the initial number of agents is odd, the ring consists of $n = k+2$ nodes and there are initially k agents and one black virus. Each node is initially occupied by one agent, except the node that contains the black virus and one of the nodes incident to the black virus (see Figure 4.7). In either case, each agent has the same number of tokens and neighbouring agents are forced by the adversary to move in different directions: agent A_1 moves clockwise and agent A_{i+1} moves opposite than A_i, $\forall i \geq 1$.

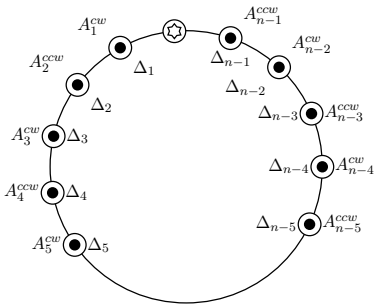

FIGURE 4.6: An initial configuration consisting of $n-1$ anonymous agents, where $n-1$ is an even number.

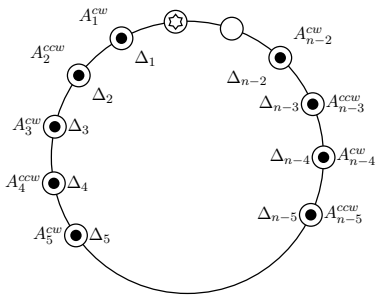

FIGURE 4.7: An initial configuration consisting of $n-2$ anonymous agents, where $n-2$ is an odd number.

If the initial number of agents is even, two agents, A_1 and A_{n-1}, along with their tokens are destroyed after the first step. If the initial number of agents is odd, one agent, A_1, along with its tokens is destroyed after the first step. In both cases, each of the remaining agents reaches a node containing the same number of tokens. The configuration is that of Figure 4.8.

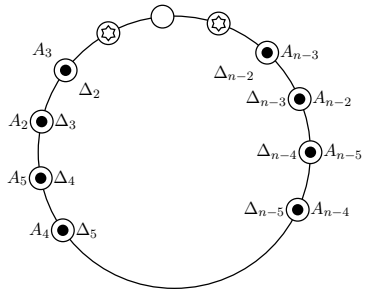

FIGURE 4.8: The resulting configuration after the agents' first move in configuration of Figure 4.6 or Figure 4.7.

The agents cannot stop moving after finding a token, otherwise, the ring will not be cleaned. If the agents continue moving in the same direction, two additional agents and two piles of tokens are lost immediately and once again all the remaining agents reach the same number of tokens. If the agents change direction and move until they find a token, all agents return to their respective homebases and no agent is destroyed. However, the agents cannot keep moving between any two nodes with tokens indefinitely, otherwise the ring will not be decontaminated. Therefore, the agents eventually have to move to a node they have not visited before. Consequently, in any case two additional agents and two piles of tokens are destroyed, leaving $k-4$ agents, $k-4$ nodes with (the same number of) tokens and five nodes that need to be cleaned. The configuration is shown in Figure 4.9.

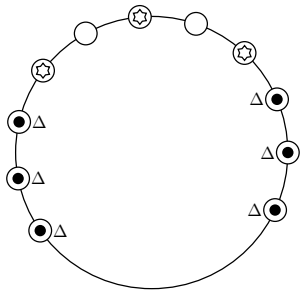

FIGURE 4.9: The resulting configuration after some moves of agents in the configuration of Figure 4.8

The configuration of Figure 4.9 is similar to that of Figure 4.8, with the difference that there are two more nodes that need to be cleaned and two agents less. The agents on Figure 4.9 do not have any further knowledge of the network and cannot move in a more effective / different way than the agents of Figure 4.8. More generally, the ring cannot be cleaned unless the agents visit new nodes. However, each time the agents move to a new node, the two agents close to a node containing a black virus are destroyed along

with their tokens. Each time an agent is lost, a pile of tokens is lost as well. Thus, any two agents have always the same input and therefore they cannot break this symmetry and they all eventually vanish. □

4.2.2 An algorithm with twelve agents

We present here an algorithm for twelve agents with distinct identities and one token each which can decontaminate any unoriented synchronous ring. We use the following notation.

$$f \in \{cw, ccw\}$$

The direction f is chosen by the adversary to be either cw (i.e., clockwise) or ccw (i.e., counter-clockwise), with the agent not having knowledge of that. We define f' as the opposite direction of f. The agents start by releasing their tokens at their initial locations, and move one step forward (either cw or ccw whichever the adversary chooses). Then the agents that survived move back to pick up their respective tokens and move forward to the safe node they explored and release their tokens there. Afterwards, the agents start moving towards the direction opposite to the direction they moved on their first move trying to converge with another agent onto the same node.

During this phase, the agents bounce back and forth between the two tokens they find, bringing those tokens closer and closer (by moving only their token). Eventually, at least two agents meet and start to move trying to decontaminate the ring using Procedure `Cautious-Move`.

Algorithm 6: ClearRing

```
1 release token
2 move one step towards f
3 execute Secure
4 if did not meet agent during Secure then
5 |   execute Converge
6 |   repeat
7 |   |   wait
8 |   until meet with another agent carrying a token OR meet a pair
9 execute Decontaminate
```

Procedure Secure

1 move one step towards f'
2 pick up token
3 move one step towards f
4 release token

Procedure Bounce

1 **repeat**
2 | move one step towards f'
3 **until** *token found*
4 **repeat**
5 | move one step towards f
6 **until** *token found*

Lemma 4.11. *At most four agents are lost until all agents have finished executing Procedure Converge. Furthermore, the contaminated interval has been surrounded by tokens and all remaining agents are located in the part of the ring divided by those tokens and not containing the black virus.*

Proof. In the first two lines of Algorithm 6, each agent leaves its token at its initial node and moves one step towards direction f. After the move, there are $c \geq 10$ agents still alive, and $c' \geq 10$ tokens.

The remaining agents proceed to line 3. In Procedure Secure the agents first move back to their starting node to pick up their token. If $c > c'$ then in the previous phase two tokens and one agent were lost. Notice that at least one agent must vanish in order for a token to disappear. The agent that has no token, due to the black virus, is destroyed trying to return back to its token. We still have $c \geq 10$ and $c' \geq 10$ but now $c = c'$.

After line 3 of Algorithm 6 each remaining token must be at least two edges away from any black virus towards f'. Thus, any agent moving towards f' does not result in the loss of any more tokens, but it may result in the loss of more agents. More specifically, the agents that can be destroyed are the two agents located closest to both endpoints of the contaminated interval since all other agents reach a node containing a token. □

Lemma 4.12. *After a finite number of moves, at least two agents meet.*

Procedure Converge

```
repeat
    execute Bounce
    if token is one step away then
        execute Check
    else
        pick up token
        move one step towards f'
        release token
until another token is on same node
```

Procedure Check

```
Let x be your identity number
wait for x time units
pick up token
move one step towards f'
```

Proof. In view of the previous lemma, by the time the agents finish the execution of the Procedure Converge, the contaminated interval is surrounded by two tokens and the part of the ring divided by those tokens and including the black virus does not contain any of the at least 8 remaining agents. Let p_1 and p_2 be those tokens and let, without loss of generality, p_2 be the token clockwise from the endpoint of the contaminated interval. There are two distinct cases:

1. Each surviving agent A_i executing Procedure Converge has the same sense of direction as any other surviving agent A_j executing the same procedure (i.e., $f_{A_i} = f_{A_j} \forall i \neq j$). Since all remaining agents have a common sense of direction either p_1 or p_2 (or both) should have been initially belonged to an agent that vanished. Since none of the remaining agents move past a token left by another agent, at some time t_1, an agent A will move its token to a node with a token left by the agent that was lost. Agent A, stops and waits until another agent reaches the same node. Since all agents move in the same direction, another agent A' will reach the same node as A at some time $t_2 > t_1$.

2. There are at least two consecutive agents A'_1, A'_2 that move in opposite directions when executing the same command (i.e., $f_{A'_1} = f'_{A'_2}$). If there are more than two such agents, we choose A'_1 and A'_2 to be two consecutive agents that move towards

Procedure Decontaminate

```
1  agent with biggest id becomes leader and all others become companions
2  guard_found ← false
3  repeat
4      execute Cautious-Move(dir_of_leader)
5      if meet agent performing guarding action then
6          if guard_found = false then
7              change direction
8              guard_found ← true
9          else if guard_found = true then
10             guarding agent becomes companion and executes
                Cautious-Move(dir_of_leader)
11     else if meet another pair then
12         agent with biggest id becomes leader and all others companions
13 until leader lost
14 if number of companions on node = 1 then
15     begin guarding action
16 else
17     agent with biggest id becomes leader and all others companions
18     if guard_found = false then
19         agent with smallest id begin guarding action
20         all other agents on node change direction
21 repeat
22     execute Cautious-Move(dir_of_leader)
23     if leader not on node then agent with biggest id becomes leader
24 until ring is clean
```

each other if such a pair exists. If there is originally an odd number of nodes between A'_1 and A'_2 eventually the two agents simultaneously move their tokens to the same node v, where the distance $dist\{A'_1, v\} = dist\{A'_2, v\}$ at any time. If there is originally an even number of nodes between the two agents, the agents eventually place their tokens on neighbouring nodes and detect that there is an agent on a neighbouring node (Procedure Check). After detecting another agent and a token on a neighbouring node, the agent returns to the node where it left its token and waits for a number x of time units, where x is the agent's identity label. The agent that finishes *waiting* first, moves one step towards f' and the two agents meet. Let us now consider the case in which A'_1 and A'_2 move away from each other. In this case, all the remaining agents in the ring are divided into two groups. One of the groups consists only of consecutive agents that move in the same way as A'_1, while the other group consists only of consecutive agents that move in the same way as

A'_2. Any one of the groups can possibly contain only a single agent (A'_1 or A'_2 respectively). The agents in the group or the groups that contain more than one agents will form at least one pair in the way described in case 1 of this lemma.

□

Lemma 4.13. *Twelve agents with distinct identities (taken from the set of natural numbers), constant memory and one token each can decontaminate any $n-$node, synchronous, unoriented ring using Algorithm 6 within $O(n^2 + L)$ time units, where L is the maximum identity label.*

Proof. As we showed in Lemma 4.11, at most four agents are lost until all agents have finished executing Procedure Converge. The first agent that is lost creates a contaminated interval of three nodes. Each subsequent agent that is lost increases the size of the contaminated interval by one node. Therefore until the agents start executing Procedure Decontaminate, the contaminated interval consists of at most six nodes. As we showed in Lemma 4.12, at least two agents will meet after a finite number of steps and will therefore start executing Procedure Decontaminate. When executing Procedure Decontaminate, the agents only move using Procedure `Cautious-Move`. When a pair meets another pair or team, all agents form a single team and the agent with the biggest ID becomes the leader. When a team of agents encounters for the first time an agent that is guarding the contaminated interval, the team changes direction. Notice that since at least one team of agents reaches one end of the contaminated interval, changes direction and approaches the contaminated interval from the opposite direction all teams in the ring will eventually merge. When a leader is lost, its companion begins guarding the node incident to the contaminated interval until a team that has already met another guard reaches it. Therefore, when the agents clean more than one nodes in the same direction, both ends of the contaminated interval are guarded and the black virus cannot spread to new nodes. Finally, the eight remaining agents clean the six nodes in the contaminated interval in the following way. Five agents clean five of the nodes in the contaminated interval by moving using Procedure `Cautious-Move`. One black virus and three agents remain. When one of the agents moves to the infected node, both of the nodes incident to the last contaminated node are occupied by one companion and an

agent performing a guarding action and the ring is cleaned. It is easy to see that the algorithm takes at most $O(n^2 + L)$ time units (due to Procedure Converge). □

4.2.3 An algorithm for six agents provided with an advice

As we proved in Lemma 4.3, five agents with distinct identities and unlimited tokens cannot solve the problem in oriented rings even when the agents know their initial configuration. We show here that if the agents are provided with an advice that encodes a few agents' identities then the problem can be solved in any unoriented ring by only six agents without tokens.

Let $L = \{A_1, \ldots, A_6\}$ be a sequence of 6 successive agents, where $\forall i : 1 \leq i \leq 6$, A_i are their distinct identities (which are taken from a totally ordered set, e.g., positive integers), A_1 is the minimum one and A_2 is the minimum identity between the identities of the two successive agents (clockwise and counter-clockwise) of A_1. An example is shown in Figure 4.10. If an agent's identity appears at an odd index in this sequence, then the agent will be instructed to move, otherwise it will be instructed to wait. Hence, exactly 3 of the agents in the example will be instructed to move. We describe the algorithm below.

All agents receive the same sequence with the identities of the agents that are instructed to move as advice (which needs $O(\log L)$ space, where L is the maximum identity label), and they move until they meet an agent. If a group of agents is formed then the agents switch to state 'group1' and they start to move in the opposite direction of the moving agent's direction using Procedure `Cautious-Move`. If a group of two or three agents at state 'group1' meets the Black Virus then the surviving agents switch to state 'guard' and wait until they meet an agent. If a non-group agent meets a guard then it switches to state 'explorer', it changes direction and moves until it meets an agent. If a group of agents at state 'group1' meets a guard then the agents of the group switch to state 'group2', they change direction and move using Procedure `Cautious-Move`. If an agent meets an explorer then they form a group of agents, they switch to state 'group2', and they move in the direction of the explorer using Procedure `Cautious-Move`.

Lemma 4.14. *Six initially scattered agents with distinct identities (which are positive integers) suffice to decontaminate any, synchronous unoriented ring, if they have an initial*

Algorithm 7: AdviceProtocol

Input: $L = \{A_1, \ldots, A_6\}$

```
if identity at odd index then
    state ← moving
    repeat move one step f until met with an agent
    if agent met is a stationary then state ← group1
    else if agent is from a group then follow group
    else if there is exactly one guard then
        state ← explorer
        repeat move one step f' until met with an agent
        state ← group2
    else state ← group2
    execute Group
else
    state ← stationary
    repeat wait until met with an agent
    if met with a moving agent then state ← group1
    else if met with an explorer then state ← group2
    else follow group
    execute Group
```

advice of size $O(\log L)$, where L is the maximum of the agents' identities.

Proof. Notice that each moving agent is between two stationary agents, denoted by S_1 and S_2. Since there is one black virus initially, at most one agent is lost before an agent meets another agent and a contaminated interval of at most 3 nodes is produced. For example in Figure 4.10 only agent A_5 could meet the black-virus and vanish before a group of at least two agents is formed.

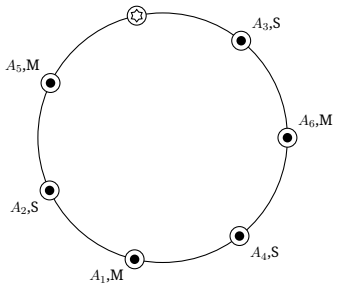

FIGURE 4.10: A proper sequence for this initial configuration is $L = \{A_1, A_2, A_5, A_3, A_6, A_4\}$. 'S' denotes a *stationary agent* and 'M' a *moving agent.*

The first group (group A) that is formed starts to move in the opposite direction of the moving agent's direction using Procedure `Cautious-Move`. If a non-group agent, or a

Procedure Group

```
1  switch state do
2      case group1 do
3          repeat execute Cautious-move(f′) until BV is found ∨ met with a guard
4          if BV is found ∧ group had two or three agents then
5              state ← guard
6              execute Guard
7          else if BV is found then
8              if have lowest identity then
9                  state ← guard
10                 execute Guard
11             else
12                 state ← group2
13                 execute Group
14         else if met with a guard then
15             state ← group2
16             execute Group
17     case group2 do
18         repeat execute Cautious-move(opposite of previous group's direction) until
            ring is decontaminated
```

group of agents reaches a guard then it changes direction. If a group of three agents reaches the Black Virus then the remaining two agents wait (as guards).

Since each moving agent is between two stationary agents S_1, S_2, if the group is formed at S_1 then it moves towards S_2 and vice versa. Suppose, without loss of generality, that the group A is formed at S_1 and moves towards S_2.

There are three possible cases.

1. Group A reaches S_2, and the group now has three agents
2. There is a black virus between group A and S_2, thus group A reaches the virus and its leader is destroyed making the other agent a guard
3. S_2 becomes part of another group (group B) and that group starts moving the same direction as group A

We examine each case separately. From here on we refer to the initially stationary agents as **S**, the initially moving agents as **M**, and the guards as **G**.

Procedure Guard

```
1 repeat wait until met with any agent
2 if met with a group2 ∨ explorer ∨ ((moving agent ∨ group1) ∧ there are two guards)
    then
3     if there are two guards ∧ have the lowest identity then
4         state ← guard
5         execute Guard
6     else
7         state ← group2
8         execute Group
```

Case 1. In this case we have a group of 3 agents (two **S** and one **M**). There are 4 sub-cases. Either the group reaches a Black Virus, or the group meets with a **M**, or the group meets with another group, or the group meets a **G**. Lets examine those more carefully.

1. The group (of three agents) reaches a Black Virus.

   (a) If this is the first time **any agent** reaches a Black Virus, then the leader of the group vanishes and the Black Virus spreads towards the unguarded neighbor. We now have one Black Virus and 5 agents, of which two **G**s, two **M**s, and one **S**.

   Firstly, if a second group reaches the Black Virus then the ring is decontaminated. Secondly, if a moving agent reaches the guards, then one guard and the moving agent form a group which now moves in the opposite direction collecting the stationary agent and decontaminating the ring, even if the other moving agent reaches the Black Virus first. Thirdly, if a group reaches the guards, then one guard is collected and the group, with either three or four agents, moves in the opposite direction decontaminating the ring.

   (b) If this is the first time **a group** reaches a Black Virus, then a moving agent has already vanished before, making it a contaminated interval of three nodes. The leader vanishes and the Black Virus does not spread further. We now have a contaminated interval of 2 nodes and 4 agents, of which two **G**s, one **S**, and one **M**. The moving agent cannot reach the Black Virus before meeting another agent and therefore either meets the guards or the stationary agent. In both cases it forms a group. If the group reaches the guards, then one guard

is collected and the group of three agents move in the opposite direction decontaminating the ring. If the moving agent firstly meets the guards, then a group is formed which collects the stationary agent decontaminating the ring.

(c) If this is the second time a group reaches the Black Virus then the ring has been decontaminated.

2. The group (of three agents) meets with a **M**. Then there is one **S** and one more **M** left. The group can then reach either the Black Virus, or another group, or **S**. If the group reaches the Black Virus after the other group then we are done. If it reaches the Black Virus first then one agent vanishes, one becomes a guard, and the other two form a group moving in the opposite direction decontaminating the ring. If two groups are met then all agents have gathered and can decontaminate the ring. Lastly, if the group collects **S** then even if the remaining **M** vanishes before it is collected there is a contaminated interval of 3 nodes and a group of 5 agents which can decontaminate the ring.

3. The group (of three agents) meets with another group. Then we have a group of 5 agents which can decontaminate a contaminated interval of 3 nodes.

4. The group (of three agents) meets a guard. Then it changes direction collecting any redundant agent and decontaminating the ring.

Case 2. If the group reaches the Black Virus, then this is either the first time any agent reaches the Black Virus, or this is the first time a group reaches the Black Virus, or this is the second time a group reaches the Black Virus, hence the ring is decontaminated. We examine the first two cases more closely.

1. If it is the first time any agent reaches the Black Virus then we have a contaminated interval of one node and 5 agents left. The agents left are one guard, 2 **M**s, and 2 **S**s. This is a similar case to case's 1.1.a and can be examined in a similar manner.

2. If it is the first time a group reaches the Black Virus, then a moving agent has reached the Black Virus before. The group reaches the Black Virus and we now have a contaminated interval of 2 nodes, one guard, 2 **S**s, and one **M** which cannot reach the Black Virus before meeting another agent. Then it either meets the guard

and changes direction, or it meets with a **S** forming a group. In either case a similar argument as before can be presented.

Case 3. Since group A never meets S_2, it can only meet with group B or meet a guard. If the two groups meet then there is one **S** still left and either a **M** depending on the number of agents of group B. If group B has three agents, then 5 out of 6 agents have met and they can decontaminate the ring collecting the remaining **S**. If group B has only two agents then 4 agents have gathered and there is still a **M** moving. Even if the last moving agent reaches the Black Virus first the ring can be decontaminated by the rest agents 4 in a group and one stationary, because the contaminated interval does not increase further.

In all three cases the ring is decontaminated. □

4.3 Discussion

We can show that the algorithm for unoriented rings (with twelve agents) will decontaminate any oriented ring using at most ten agents (i.e., at most nine will vanish). Hence in other words, the algorithm for unoriented rings is a uniform algorithm that is optimal with respect to the number of agents in both oriented or unoriented rings. However, it takes $O(n^2+L)$ time units to finish even in oriented rings, where L is the maximum label of the agents, in contrast with the algorithm for oriented rings which takes $O(n)$ time units to solve the problem with ten agents. Whether there exists an $O(n)$ algorithm for unoriented rings is an interesting open question.

Another difference between the two algorithms is the following. The algorithm for oriented rings can work for anonymous agents mainly because the agents use their distinct identities to assign different roles to themselves only when they are co-located. However, the algorithm for unoriented rings heavily relies on agents' distinct identities: they are distinct positive integers and each agent might need to use its label even before meeting another agent. In fact as we proved, the problem is unsolvable by any number of anonymous agents in unoriented networks.

Another interesting question is whether those algorithms can be extended to handle asynchronous networks. Does there exist an algorithm for unoriented rings using eleven agents with more than one token each? Finally it is interesting to investigate whether

there are different (or better) solutions for rings consisting of n nodes, where n is small. For example it is easy to see that if $n \leq 3$ the problem is unsolvable due to Lemma 4.1 (even for agents with advice). If $n = 4$ it is solvable by 3 agents with advice, but otherwise impossible. If $5 \leq n \leq 7$ then 4 agents with advice can solve the problem.

5 Broadcasting

The communication between nodes of a network has been studied a lot in the literature. The classical problems of broadcast or convergecast deal with the dissemination of information in the network. In the case of message passing networks, broadcast is achieved by spreading the information from the source node to all other nodes. For a system of mobile agents, the equivalent problem is the propagation of information from one source agent to all other agents in the system. Such problems are relevant for teams of mobile sensor robots sent on data collection missions. We assume that the agents autonomously move along the edges of a graph that represents the network; when two agents are at the same vertex, they can communicate and share information. However, the agents cannot communicate from a distance.

The problem of broadcasting has been originally investigated in message passing multihop radio networks (e.g. see the survey [53]). Previous studies on broadcast and other communication problems have focused on the efficiency of performing the task, either in terms of time taken [22], or in terms of energy expended [26]. A slightly different line of research considers the problem of communicating in the presence of faults and the objective is to tolerate as many faults as possible. The faults can be missing links or nodes [54] in the network or loss of messages [55], in case of message passing networks. Recently there has been a lot of interest in so called *dynamic networks* which model both faults and changes in network topology in a uniform manner by considering that the network may change in each round during the execution of the algorithm. As discussed in 2.1.3 the *evolving graph* model represents a dynamic network by a sequence of graphs $\mathcal{G} = G_1, G_2, \ldots$ based on the same set of vertices V but the set of edges changes in each round i, i.e. each graph $G_i = (V, E_i)$ is a spanning subgraph of the graph $G = (V, \cup E_i)$, which is called the footprint of the dynamic network. For solving most problems, some

assumptions about the connectivity of the dynamic network need to be made. We consider the model of *constantly connected* dynamic networks where in round i, the graph G_i is assumed to be connected. No other assumptions are made about the network. This means that in the worst case, the graphs G_i and G_{i+1} in two consecutive rounds may differ completely in the set of available edges. To show correctness of our algorithms, we will assume that an adversary having knowledge of the algorithm, chooses the graph G_i in each round (respecting the connectivity constraint). Note that these assumptions are much weaker compared to that of T-interval connected networks as in several previous results [60, 64] where the network is assumed to contain a stable spanning tree for a continuous period of T rounds.

Following the definition of the problem which appeared in Section 2.4.3 let us give some more details about this study. We consider different graph families, including sparse graphs, as well as dense graphs, for example when G is a ring or a cactus, or when G is a grid, a hypercube or a complete graph respectively. For each family we present algorithms in order to solve the broadcast problem using the minimum number k of agents. Note that, having more agents makes the problem easier, due to the fact that when at least one of them reaches a source agent, there is progress in the propagation of the message. We also show strict lower bounds on the minimum number of agents needed to solve the problem in the graph families mentioned earlier. Preliminary results of this chapter have appeared in [37]

5.1 Broadcast model

In this section the capabilities of the agents and the adversary model is defined.

5.1.1 Agents

The agents are autonomous and identical entities, having their own internal memory, and are able to move along the available edges of the graph. The agents cannot mark the nodes or edges of the graph. The agents are initially located at distinct nodes of the network, they all start in the same initial state, and they execute the same deterministic algorithm. They have global visibility and they move in *synchronous* steps, i.e., time is

discretized into atomic time units called rounds. During each round r, each agent can see the graph G_r and the location of all agents in G_r and can distinguish which agents have the message $\mathcal{M}$ (called *source agents*), and which agents do not (called *ignorant agents*). Based on this information, the agent can decide to stay at the current node or move to a neighboring node in G_r. In the latter case, the agent arrives at its destination node at the end of the round. At the start of the next round $r + 1$, the adversary chooses the graph G_{r+1}, and the agents execute the next step of the algorithm. In the initial round, there is exactly one source agent and k ignorant agents in the network.

5.1.2 Adversarial model

The adversary can decide the initial placement of all the agents in the network, and in each round the adversary chooses the graph G_r which represents the available links in the network for that round. The adversary may have knowledge of the algorithm and can use this knowledge for deciding the placement of agents and the dynamicity of the network (subject to the connectivity constraint as described).

Unknown Adversary: The agents do not have any knowledge of the adversary, and thus they do not know the dynamic network $\mathcal{G}$ in advance.

Global visibility: We assume global visibility to simplify the task of achieving broadcast. Note that, if the agents are restricted to local visibility in an unknown dynamic network, even in the simplest case of a ring network, it may not be possible for two agents to meet, and thus broadcasting would not be possible.

Distinct starting locations: We assume the agents are initially located at distinct nodes of the network.

5.2 Preliminaries

We make some preliminary observations about the problem. Recall that the dynamicity of the network is unknown to the agents. However, in each round i the agents can see the graph G_i and the positions of all agents in it.

Notice that, due to the facts that the agents start at distinct locations, they have global visibility, there is a unique source agent and the network is locally oriented, the agents can assign themselves distinct identities in the following way:

Each agent computes for every agent A_i (including itself) initially located at a node u_i the lexicographically minimum string of port labels that corresponds to a shortest path from node u_i to node v where the source agent is initially located. Hence all agents have constructed the same list of strings. Further they can lexicographically sort in increasing order those strings and assign accordingly distinct identities to all agents. Once the identifiers are computed in a pre-computation, the agents can remember these identifiers and can track all other agents during the algorithm (thanks to the global visibility). Hence, in the following, we assume without loss of generality, that all agents have distinct identities. Moreover, for simplicity, we shall describe the algorithms in a centralized manner, describing which agents perform which operations in any round. It is evident that the agents executing the same algorithm, can autonomously decide their role in the computation. Finally, we denote with k the number of ignorant agents (i.e., the agents that do not know the initial message).

Observation 1. *Given a constantly connected dynamic network $\mathcal{G}$ based on an arbitrary connected graph G consisting of n nodes, with $k \geq n-2$ ignorant agents on distinct nodes of the network, then within $O(k)$ steps, we can solve the broadcast problem.*

Proof. Since there are at least $n-2$ ignorant agents on distinct nodes and one node is occupied by the source agent, there is at most one empty node. Thus, in any connected graph G_i there would be a path of length at most two between a source agent and an ignorant agent. These two agents would meet in this round. So, we reduced the number of ignorant agents. The agents can now spread to distinct nodes (using their distinct identities that have been assigned before they meet, as described in the previous subsection), and we can repeat the same argument. □

The above result provides a general upper bound on the team size needed for solving broadcast. We will present smaller bounds for specific graph topologies. For the special case of trees, the adversary cannot block any edge without losing connectivity. Hence we have the following trivial result for trees.

Observation 2. *If G is a tree then broadcast can be solved for any $k \geq 1$.*

Proof. The adversary cannot block any edge without disconnecting the graph. Thus, the graph is static and in each step each agent can move one step closer to the node containing the source agent, thus in $O(D)$ time all agents would be colocated with the source agent and we solve broadcast in $O(D)$ time, where D is the distance of the farthest agent from the source. □

5.3 Broadcast in sparse graphs

In this section, we will study the broadcast problem with agents in sparse graphs. A simplest non-trivial sparse network is the ring (or cyclic graph 2.1) topology.

Theorem 5.1. *If G is a ring of size $n \geq 5$, then broadcast can be solved if and only if $k \geq 2$ within $O(n)$ steps. If G is a ring of size $n < 5$, then broadcast can be solved for any $k \geq 1$ within at most 2 steps.*

Proof. Consider a ring of size $n \geq 5$, with one source that has the information $\mathcal{M}$ and one ignorant agent. At each step, the adversary can remove an edge on the shortest path between the two agents. Note that, the longer path is always of size at least 3, thus the agents cannot meet on this path in one step. Hence the two agents can never meet.

Now we show that if there are at least two ignorant agents ($k \geq 2$), then broadcast is possible. The two agents can try to reach the source agent by opposite directions, then at each step one of the agents gets closer, and eventually one of the agents would reach the source and obtain $\mathcal{M}$. At this stage there are 2 source agents, they can traverse the ring in opposite directions, thus at least one of them will soon meet the remaining ignorant agent and broadcast is solved within $O(n)$ steps.

The impossibility of solving broadcast with $k = 1$, does not hold for rings of size 3 as in this case any path between the source and the other agent is of size at most 2, and since one of these paths must be available, the two agents can meet in one step and solve the problem. For rings of size 4, if the longer path between the source agent and an ignorant agent has length 3, then one of the agents moves so that both paths between the agents are of length 2. Then within the next step, the two agents meet in one of those paths,

similarly as in the case of rings of size 3. Thus, broadcast is solvable for rings of size $n < 5$ with any $k \geq 1$. □

We now make the following observation that will allow us to generalize the results from rings to other graphs containing cycles.

Lemma 5.1. *If G contains a cycle C of length at least 3, such that there is a single node $v \in C$ that is connected to nodes in $G \setminus C$, then the adversary can always prevent at least one agent located in $C \setminus v$ from reaching node v, thus trapping the agent in cycle C.*

Proof. Consider an agent located at a node $u \in C \setminus v$. Since the cycle must be of length at least 3, the longer path from node u to node v is of length ≥ 2. If the adversary always blocks the shorter path from the agent's location to node v, then the agent cannot reach node v. The only way to get out of the cycle is passing through node v, so the agent is forever trapped in C. □

Lemma 5.2. *If G is a ring of size $n \geq 3$, given any vertex $v \in G$, if there are two agents at distinct vertices of G, then there is an algorithm to ensure that within $O(n)$ steps either (i) the two agents meet at a vertex of G or (ii) at least one of the agents (chosen by the algorithm) can reach vertex v.*

Proof. Let us call the two agents A and B. If we require agent A to reach node v, then the algorithm asks agent B to move along the path containing agent A and then node v in this order (if agent B was already at node v then we first move it to any neighbouring node of v). In each round, only one edge of G may be unavailable, so either agent B will move closer to agent A or agent A will move closer to node v. So eventually either condition (i) or (ii) will be true. □

In the following we consider cactus graphs which can be seen as combinations of trees and rings.

Definition 5.1. *A* cactus graph *is a connected graph in which any two simple cycles have at most one vertex in common.*

In cactus graphs, the size of the team depends on the number and sizes of the cycles of the graph, as follows:

Lemma 5.3. *If G is a cactus graph of size n having $c_1 \geq 1$ cycles of length < 5 and no larger cycles, then broadcast can be solved if and only if $k \geq c_1$ within $O(n)$ steps.*

Proof. Consider the family of cactus graphs obtained from a line of length c_1 by attaching to each vertex of the line a cycle of length < 5. Each cycle thus satisfies the conditions of Lemma 5.1.

If $k < c_1$, the total number of agents is $k+1 \leq c_1$, so the adversary can place each agent in a distinct cycle, including the source agent. No agent can leave its cycle due to Lemma 5.1. So, no two agents can meet, thus broadcast is not possible.

For $k \geq c_1$, broadcast is solvable in any cactus graph with c_1 small cycles. To prove this it is enough to analyze the case where at least one cycle has at least two agents, either two ignorant ones or one ignorant agent and a source. This is because if an agent is outside of any cycle it can always move to a cycle (all non cyclic edges are available in each round). If two agents are in a cycle and none of them is the source, then one of the agents can leave the cycle within at most 2 steps (both agents move towards an elected exit approaching it from different directions). The agent that leaves the cycle can move towards the source, until it reaches another cycle.

Thus, one agent will eventually reach the cycle containing the source. This agent can meet the source and obtain the information $\mathcal{M}$ due to Theorem 5.1. Now, there are two source agents in the same cycle, and thus, one of them can leave the cycle as described before within at most 3 steps. This source agent reaches another ring containing an ignorant agent, the information is propagated and we have again two source agents in a cycle. Repeating the same algorithm, all agents will eventually learn the information, and thus we can solve broadcast within $O(n)$ steps. □

Lemma 5.4. *If G is a cactus graph of size n having c_2 cycles of length ≥ 5 and no cycles of smaller length, then broadcast can be solved if and only if $k \geq c_2 + 1$ within $O(n)$ steps.*

Proof. Consider the family of cactus graphs obtained from a line of length c_2 by attaching to each vertex of the line a cycle of length ≥ 5.

Suppose that $k < c_2 + 1$. Then, the adversary places each of the $k \leq c_2$ agents in a distinct cycle and the source in one of these cycles. Due to Theorem 5.1, the source and the other

agent cannot meet, since the ring is of length ≥ 5. At the same time, due to Lemma 5.1 the adversary can trap each other agent in its cycle. In the cycle that contains the source, at most one of the two agents can exit this cycle. Even if the source agent exits this cycle and enters another cycle the configuration is similar to the initial one. Hence, no two agents can ever meet and therefore the problem is unsolvable.

To prove that $k \geq c_2 + 1$ agents are enough to solve broadcast, we first show that at least one ignorant agent can reach the source and can become a new source. As in the proof of Lemma 5.3, we assume all agents move to some cycle if they are not in a cycle. If the source agent is in the same cycle with at least two ignorant agents, by Theorem 5.1 both these agents can become sources. If not, then, given $k \geq c_2 + 1$, there must be some cycle with two or more ignorant agents and all except one of these agents can leave the cycle to reach another cycle. Eventually two or more ignorant agents would reach the same cycle as the source, and again applying Theorem 5.1, all these agents would become source agents. Thus we have now at least $x \geq 3$ source agents in a cycle. Furthermore each of the remaining $k - x + 1$ ignorant agents are alone in some cycle. This implies that the number of empty cycles (cycles without any agent) are at most $x - 3$. Among the x source agents, $x - 1$ of them can move to another cycle. When the source agents move to an empty cycle, at most one of them may be trapped. In total $x - 3$ source agents can be trapped, thus at least two source agents can reach any cycle that contains an ignorant agent, so this agent will meet a source. Hence, all ignorant agents will eventually become sources and thus broadcast can be solved within $O(n)$ steps. □

Theorem 5.2. *Let G be a cactus graph of size n having c_1 cycles of length < 5 and c_2 cycles of length ≥ 5, then:*

- *If $c_2 = 0$, broadcast can be solved if and only if $k \geq c_1$ within $O(n)$ steps.*
- *If $c_2 > 0$, broadcast can be solved if and only if $k \geq c_1 + c_2 + 1$ within $O(n)$ steps.*

Proof. If $c_2 = 0$, then the cactus graph has only c_1 cycles of length < 5, and in view of Lemma 5.3 broadcast can be solved if and only if $k \geq c_1$.

On the other hand if $c_2 > 0$ and $c_1 = 0$, then in view of Lemma 5.4 broadcast can be solved if and only if $k \geq c_2 + 1$, and thus the second condition holds.

Finally, if $c_1 > 0$ and $c_2 > 0$, similarly as in the proofs of Lemmas 5.3 and 5.4, we can construct a cactus graph from a line of length $c_1 + c_2$ by attaching a cycle of length < 5 to each of the first c_1 vertices and attaching a cycle of length ≥ 5 to each of the remaining c_2 vertices. Now, if $k \leq c_1 + c_2$, then the adversary places each ignorant agent in a distinct cycle, and places the source agent in the last big cycle C of length ≥ 5. Since there is at most one ignorant agent and the source in cycle C of length ≥ 5, they cannot meet (see the proof of Theorem 5.1). Furthermore no other agent (in a cycle different than C) can leave its cycle due to Lemma 5.1. The source agent may escape from the cycle C and reach another cycle C'. If the cycle C' is big (size ≥ 5), then as before the source agent would not be able to meet the only agent that is in cycle C'. On the other hand, if the source reaches a small cycle (size < 5) it may meet the ignorant agent in that cycle, so we will have two sources; however at most one of the two can leave this cycle. Thus the agents in the big cycles would never meet any source agent. Thus broadcast can not be solved.

We now show how to solve broadcast using $k \geq c_1 + c_2 + 1$ ignorant agents. First, as argued before, any agent that is not on a cycle can move to the nearest cycle. Since there are more agents than cycles, there are some cycles that contain multiple agents. In any such cycle, one of the agents can move to a neighboring empty cycle if there is one. Repeating this process, we can distribute the agents such that there is at least one agent in each cycle.

Let C be the cycle that contains the source. In any cycle other than C, if there are more than one ignorant agents, all except one of them can move to another cycle that is closer to cycle C. Repeating this process, we will reach a configuration where there will be at least 2 ignorant agents and the source in cycle C and exactly one agent in each other cycle. Now it is easy to solve broadcast from this configuration. Using the ring algorithm (Theorem 5.1) all ignorant agents in cycle C would become sources. Since we have at least three source agents now, at least two of them can move to a different cycle. In any other cycle reached by those two source agents, there is one ignorant agent, so we can apply the same algorithm and have 3 source agents in this cycle. Repeating this process all ignorant agents will become sources and broadcast is solved within $O(n)$ steps. □

5.4 Broadcast in Grids

We now study grid graphs which are slightly more dense than rings or cactuses. Even for 2-dimensional grids, we show that we need $\Omega(n)$ agents to solve broadcast. Let us first consider the simplest grid graph with only two rows (called ladder graph).

Lemma 5.5. *If G is $2 \times L$ grid graph, then broadcast is unsolvable for $k < L - 1$.*

Proof. Suppose that $k < L-1$. The adversary can put all k agents in one row consisting of L vertices and the source agent in the other row. So, there is at least one column where both vertices are empty. The adversary would allow this edge and remove all other edges connecting the two rows. So the agents could only move within their respective rows. After some agents move, there would again be some column containing only empty vertices. So the above argument can be repeated. Thus, no agent can leave its respective row at any step, and hence no agent can meet the source. □

The above lower bound is almost tight as we can show an upper bound of $k \geq L$ for broadcast in any $2 \times L$ grid graph.

Theorem 5.3. *If G is a $2 \times L$ grid graph, then broadcast is solvable for $k \geq L$.*

Proof. We will denote the bottom left vertex in the grid as origin $[0, 0]$, and the rest of the vertices will be denoted accordingly as $[i, j]$ where is $i = 0$ for the bottom row and $i = 1$ for the top row, while $0 \leq j \leq L - 1$ is the column number. We will prove the above theorem using the following lemmas:

Lemma 5.6. *If G is $2 \times L$ grid graph containing a source agent at origin $[0, 0]$, such that the number of vertices occupied by ignorant agents is strictly greater than the number of unoccupied vertices in G, then there exists a move of subset of the agents which maintains the ignorant agents in distinct locations and, either (i) one ignorant agent meets the source agent, or (ii) the sum of distances from the ignorant agents to the source agent in G, decreases by at least one.*

Proof. Consider the spanning subgraph G_r that is available at the current round and consider any shortest path P in this subgraph from the source to any ignorant agent. Let

$P = (u_0, u_1, \ldots u_t)$ where u_o contains the source and u_t contains some ignorant agent. We say that an edge (u_i, u_{i+1}) is *positive* if $d_G(u_0, u_{i+1}) > d_G(u_0, u_i)$ and otherwise the edge is called *negative*. Thus positive edges in P increase the distances from source and negative edges decrease the distance from source (distances are measured in the original grid G). Notice that the distance between vertices u_0 and u_t in G is strictly less than the length of P if and only if there is a negative edge in P. If there is an ignorant agent on vertex u_{i+1} and vertex u_i is empty, then moving this agent to u_i would satisfy condition (ii) of the lemma, if and only if edge (u_i, u_{i+1}) is *positive*. So, let us assume there is no such positive edge with an empty vertex on one end and an agent on the other end. In this case we need to look for a move of a sequence of consecutive agents on a path P from the source.

If the path P from the source contains a sequence of $x \geq 2$ consecutive *negative* edges, these must be preceded by at least $x + 1$ positive edges. (This corresponds to a path that makes a U-turn and goes back towards the origin, see Figure 5.1.) If path P started with these edges, then vertices at the end of each positive edge must be empty otherwise we immediately obtain a move satisfying the conditions of the lemma. In other words, if there is no move on P satisfying the lemma, then, there are more empty vertex than vertices occupied by ignorant agents on path P.

The only other possibility is that the path P starts with the sequence of 2 positive and one negative edge, followed by a repeated sequence of (3 positive, 1 negative) edges. (This corresponds to a zigzag path where the edges going down from top to bottom row are the negative edges; See Figure 5.2. The zigzag path may be followed by a path making a U-turn.) To ensure that there is no valid move of agents on P satisfying the lemma, the adversary must leave the first two vertices empty, it can place ignorant agents on the next two vertices, but must leave the next two vertices empty and so on. Thus, if there is no valid move on P satisfying the lemma, then there must be as many empty vertices as vertices occupied by ignorant agents, in P. If this condition holds for all paths in the subgraph G_r then the total number of empty vertices is not smaller than the number of vertices occupied by ignorant agents - which is a contradiction to the assumptions of the lemma. Thus the lemma holds. □

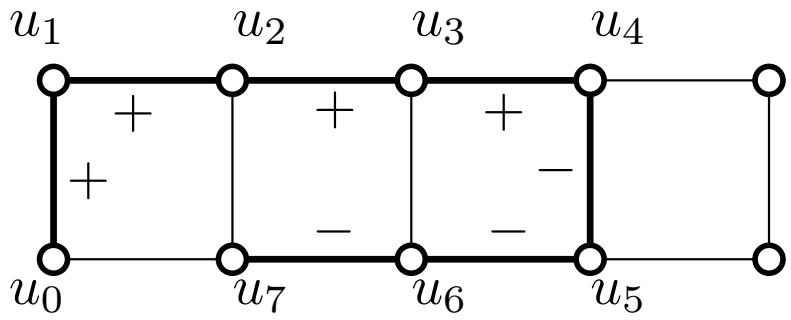

FIGURE 5.1: A path from the source making a U-turn, with a sequence of positive edge going away from the source, followed by a sequence of negative edge going toward the source.

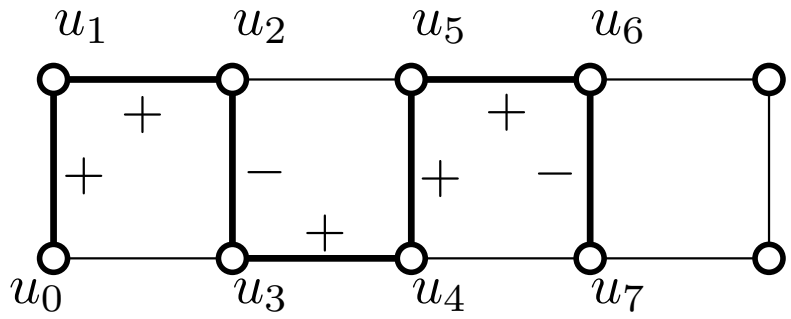

FIGURE 5.2: A path from the source that zigzags, with a repeated sequence of 2/3 positive edges followed by a single negative edge going toward the source.

Lemma 5.7. *If G is a $2 \times L$ grid graph, with $k \geq L$ ignorant agents in distinct vertices, then one of the agents meets the source.*

Proof. If G is $2 \times L$ grid graph, with the source at the origin and at least $k = L$ ignorant agents, then the number of empty vertices is $L - 1$, then we can repeatedly apply Lemma 5.6, until at least one agent meets the source. Note that the agents are always on distinct locations, until an agent meets the source.

Let us now consider the case when the source is not at $[0, 0]$ but is at a vertex $[0, j]$ (we can assume without loss that row 0 contains the source as there are only two rows). Consider the partition of G into a left grid of size $2 \times j$ and a right grid of size $2 \times (L - j)$. The algorithm would make one of the following moves:

1. If the edge $([0, j - 1], [0, j])$ is available, the source moves left to vertex $[0, j - 1]$
2. If the edge $([0, j - 1], [0, j])$ is not available, and the right grid has more ignorant agents than empty vertices, apply Lemma 5.6 on the right grid until an ignorant agent meets the source

3. Otherwise, if vertex $[1, j-1]$ is occupied by an ignorant agent, this agent moves to vertex $[1, j]$ (pushing other agents to maintain agents in distinct locations). If not, assume that vertex $[1, j-1]$ contains an "imaginary" source agent and apply Lemma 5.6 on the left grid until an ignorant agent reaches vertex $[1, j-1]$.

We first show that it is always possible to execute one of the three moves above. If the edge $([0, j-1], [0, j])$ is not available, then the edge $([1, j-1], [1, j])$ must be available. So any agent at $[1, j-1]$ can move to $[1, j]$ (first part of step (3)). Moreover edge $([1, j-1], [1, j])$ is the only connection between the left and right grids. So, each of these grids has a spanning subtree available, so we can apply the algorithm independently on one of those grids. Thus step (2) is possible. If neither of the first 2 steps are possible, then again edge $([1, j-1], [1, j])$ is the only connection between the left and right grids, plus there are more empty vertices than occupied vertices in the right grid. However, since $k \geq L$, this implies that the left grid must satisfy the condition that there are more vertices occupied by ignorant agents than empty vertices. The algorithm tries to move agents from the left to the right grid.

If the algorithm applies step (1) repeatedly, then eventually the source agent would be at the origin $[0, 0]$ or it meets an ignorant agent on the way and the lemma holds. If the source reach the origin $[0, 0]$, using Lemma 5.6 we can ensure that an agent meets the source. Now suppose step (1) cannot be applied in some round r. Consider the partition of G_r into left and right grids. As long as there are sufficiently many ignorant agents in the right grid, we can apply Lemma 5.6 repeatedly then an agent would meet the source eventually. If we apply step 3 then agents from the left grid move to the right grid (we can do so while maintaining agents on distinct locations). If enough agents move to the right grid then the conditions of Lemma 5.6 would be satisfied in the right grid, so we can apply one of the first two steps. Thus, eventually one of the ignorant agent meets the source and the lemma holds. □

Lemma 5.8. *In the algorithm described above, the ignorant agents are always in distinct locations in each round.*

Proof. Every move of an ignorant agent at a vertex v is one of the following types: (i) The neighboring vertex w contains the source the agent moves to meet the source at

vertex w. (ii) The neighboring vertex w is unoccupied and the agent moves to occupy this vertex. (iii) The neighboring vertex w contains an ignorant agent and there is a sequence of agents that move simultaneously, each agent occupying the vertex vacated by the predecessor, and the first agent in the sequence moves to an empty vertex. □

Now, we show how to solve broadcast after one of the ignorant agent has reached the source, so now there are two sources at a vertex of G. The two sources can move to adjacent columns (say columns j and $j+1$) in at most two steps. As before, we can partition the graph G_r at this round, into a left grid of size $2 \times j$ and a right grid of size $2 \times (L-j)$. We have two cases.
Case (i): Both left and right grids have a connected spanning subgraph in the current round. We can thus apply Lemma 5.6 recursively to at least one of the grids. Case (ii): One of the partitions (say, the left grid) is disconnected. Since G_r is connected this means that there are exactly 2 connected components of the left grid and these two components are connected by a path passing through the right grid. Further at least one component must be a line (i.e. a subset of a row) connected to the right grid. If there are empty vertices on the left grid, then the agents, if any, on this line can move towards the left grid (as before any sequence of consecutive agents move together). On the other hand, if the left grid has no empty vertices, we can apply Lemma 5.6 on the left grid as in case (i).

Thus we can ensure progress in each of the cases. □

Lemma 5.9. *If G is an $h \times L$ grid graph, with $h \geq 1, L > 2$, then starting from a configuration with $L-1$ agents in each row, the adversary can ensure that there are never more than $L-1$ agents in the bottom row. Moreover, if we add an additional agent at the bottom row, again the adversary can ensure that there are never more than L agents in the bottom row.*

Proof. If $h = 1$ then there is only one row, so the number of agents on the bottom row never changes and the lemma holds trivially. We now prove the lemma by induction on the number of rows. Suppose that the lemma holds for a grid G with $h \geq 1$ rows. We can construct a grid G' by adding an additional row at the bottom with L vertices and $L-1$ agents on distinct vertices. Since the bottom row of G has at most $L-1$ agents (by induction hypothesis), there exists an empty vertex v in this row. In the grid G' the

adversary makes available the edge from v to the vertex below; all other edges between G and the additional row are unavailable. In the current round, no agent can enter the bottom row of G'; although an agent from the bottom row may go up and reach grid G. In the next round if there are $L-2$ agents in the bottom row, and L agents in the row above then the adversary makes available the edge between a vertex (containing at most one agent) in this row to the vertex below in the bottom row (and disables all other edges between these two rows). So, either an agent moves to the bottom row or the number of agents in the bottom row remains the same. In the first case, we are back in the initial situation and we could use the same arguments as before. In the second case, the number of agents at the bottom row is smaller than what was initially. Thus the lemma holds for grids of $h+1$ rows and thus by induction for all grids satisfying the conditions of the lemma. Furthermore, note that all arguments remain the same if initially there are $x > L-1$ agents in the bottom row, i.e. the number of agents in the bottom row is never more than x if the higher rows contain at most $L-1$ agents initially. □

Lemma 5.10. *If G is a $W \times L$ grid graph, with $W > 2$ then broadcast in unsolvable for $k < (L-1)(W-1)$.*

Proof. We can construct a grid of size $W \times L$ by joining two grids: a grid G^1 of $(h = W-2) \times L$ with a grid G^2 of size $2 \times L$ (by adding L edges between the bottom row of G^1 and the top row of G^2). In grid G^1, we place $L-1$ ignorant agents on each row, at distinct locations, while in grid G^2 we place $k_2 < (L-1)$ agents plus the source agent, as in Lemma 5.5. Thus the total number of agents is $k = (W-2)*(L-1)+k_2 < (L-1)(W-1)$. We now show that broadcast is not solvable in this graph.

Firstly, if no additional agents enter grid G^2 and the source never leaves G^2 then by Lemma 5.5, broadcast is not possible as no agent would meet the source in G^2. Furthermore, if the source is in the bottom row, it can never leave this row and thus it cannot leave G^2. In the grid G^1 there are $L-1$ agents in the bottom row and by the Lemma 5.9 the number of agents on this row does not increase (considering only agents in G^1), so there is at least one empty vertex on this row. The edge between this vertex and the grid G^2 is made available and all other edges between the two grids are unavailable. In this

case, no agents from G^1 can enter G^2. However, some ignorant agents from G^2 can enter G^1. If x ignorant agents from G^2 enter the grid G^1, then by Lemma 5.9, the number of agents on the bottom row of G^1 is at most $L-1+x$. In each round, if there is some empty vertex v on this row, the edge between v and G^2 is the only edge between the two grids that is made available in that round (in this case, no agents can move from G^1 to G^2). Otherwise any vertex in the bottom row of G^1 can have at most x agents in this round; so, if one edge is available between the two grids, at most x agents can move from G^1 to G^2. Thus, after each round, the number of ignorant agents in G^2 is less than $L-1$ and thus broadcast is not possible by Lemma 5.5. □

We can generalize the above to higher dimensional grids as follows:

Lemma 5.11. *If G is a $d+1$-dimensional grid graph of size $W_1 \times W_2 \times \ldots W_d \times L$ where $W_i \geq L \geq 2$ then broadcast in unsolvable for $k < (L-1)(W_1 \cdot W_2 \ldots W_d - 1)$.*

Proof. Consider the subgraph of the grid that is a 2-dimensional grid of size $(W_1 \cdot W_2 \ldots W_d) \times L$. In each round the adversary chooses the available graph to be a connected subgraph of this $2D$ grid, then we can apply Lemma 5.10 to obtain the above lower bound. □

5.5 Broadcast in Dense graphs

In dense graphs there are many disjoint paths between two vertices and thus there are many possible ways for the adversary to change the network while keeping it connected. In other words, the dynamicity of a (constantly connected) dynamic graph whose footprint is a dense graph, is higher than that of sparser graphs that we studied before. The worst case is when the underlying graph is a complete graph.

5.5.1 Broadcast in Complete graphs

Lemma 5.12. *If G is a complete graph of size n then broadcast can be solved if and only if $k \geq n-2$ within $O(n)$ steps.*

Proof. If $k < n-2$ then at most $n-2$ vertices are occupied (including the source), and therefore there are at least two empty vertices. The adversary will make available the

spanning tree where the source is connected to one empty vertex and all other occupied vertices are connected to the other empty vertex. The two empty vertices are connected with an edge. This is a spanning tree of G and in this tree, the distance from the source to any occupied vertex is more than two. So no agent can meet the source in one step. After one step, during which some agents may move, there will still be at least two empty vertices; thus the same argument can be repeated for any step. Hence broadcast is impossible for $k < n - 2$.

If $k \geq n - 2$, then by the Observation 1, it is possible to solve broadcast in any arbitrary topology, and thus in a complete graph. □

The impossibility result above can be generalized to arbitrary graphs G having the following property.

Lemma 5.13. *Consider a graph G and an integer $k \geq 1$. Suppose that for every possible placement of the source agent and k agents on distinct vertices of G, there always exists a spanning tree of G where the distance from each agent to the source is ≥ 3. Then, broadcast is impossible in G with k ignorant agents.*

5.5.2 Broadcast in Hypercubes

We now study the problem in hypercube networks as defined below.

Definition 5.2. *A* d-dimensional hypercube *is a graph $H_d = (V, E)$ with $n = 2^d$ vertices labelled with distinct d-bit strings. A vertex $v_i \in V$, $0 \leq i \leq 2^d - 1$, is connected to the d vertices whose labels differ in exactly one bit from its own. Hereafter, we freely identify vertices with their labels.*

A hypercube H_d consists of two $d - 1$ dimensional hypercubes labelled as $[0 * * \cdots *]$ and $[1 * * \cdots *]$, the corresponding vertices of these two hypercubes are connected by edges of dimension d.

Theorem 5.4. *Given a hypercube H_d of dimension $d > 2$, at least $k = n/2 - 1$ ignorant agents are necessary to solve broadcast in the dynamic graph based on H_d.*

Proof. Assume $k < n/2 - 1$. Suppose the source agent is at the vertex $[00 \ldots 0]$ of H_d; the adversary places all the ignorant agents among the vertices of the sub-hypercube H_{d-1}

labelled $[1**\cdots*]$. Out of the $n/2$ vertices in $H_{d-1}[1**\cdots*]$, there must be at least 2 empty vertices. At most one of these two vertices can be a neighbor of the source vertex $[00\ldots0]$ in the other sub-hypercube. So, the other empty vertex v must be a neighbor of an empty vertex u in $H_{d-1}[0**\cdots*]$. The adversary chooses the available graph G_i as the union of a spanning tree of $H_{d-1}[0**\cdots*]$ and a spanning tree of $H_{d-1}[1**\cdots*]$, plus the edge (u,v). All other edges of dimension d are missing in G_i. After the agents move in this round, the ignorant agents would still be in sub-hypercube $H_{d-1}[1**\cdots*]$ and the source would be in the other sub-hypercube $H_{d-1}[0**\cdots*]$. Thus, using the same argument, in each round r the adversary can choose the graph G_r as a spanning tree where each ignorant agent is at a distance of at least 3 from the source, so by Lemma 5.13, broadcast is impossible. □

The above result does not hold for the trivial case of $d = 2$, since H_2 is simply a ring of four vertices where broadcast can be solved even for $k = 1$ (see Theorem 5.1). For a hypercube of dimension $d = 3$ (i.e. a cube) we can show a matching lower and upper bound of $k = n/2$ ignorant agents.

Theorem 5.5. *If $G = H_3$ is a hypercube of dimension $d = 3$ i.e. a cube consisting of $n = 2^3$ vertices, then $k = n/2$ ignorant agents are necessary and sufficient to solve broadcast.*

Proof. (**Lower Bound**) We show that if there are only $k = 3$ ignorant agents, then it is not possible to solve broadcast starting from arbitrary configurations. In particular, we define a class QF of forbidden initial configurations, with one source and 3 ignorant agents in a cube, such that starting from any such configuration, the adversary can force the agents to move only to another configuration in QF. Further, in every configuration in QF, there is a spanning tree (defined by the available links) where the distance from source to the nearest agent is at least 3. (See Figure 5.3, showing all the forbidden configurations, up to isomorphism, and the corresponding spanning trees in each case.) The configurations in the set QF are listed below by showing the positions of the 3 agents, with respect to the source vertex which is always assumed[1] to be $[000]$:

Q_1 $([100],[010],[110])$

Q_2 $([100],[010],[011])$

[1] After any moves, we rename the vertices.

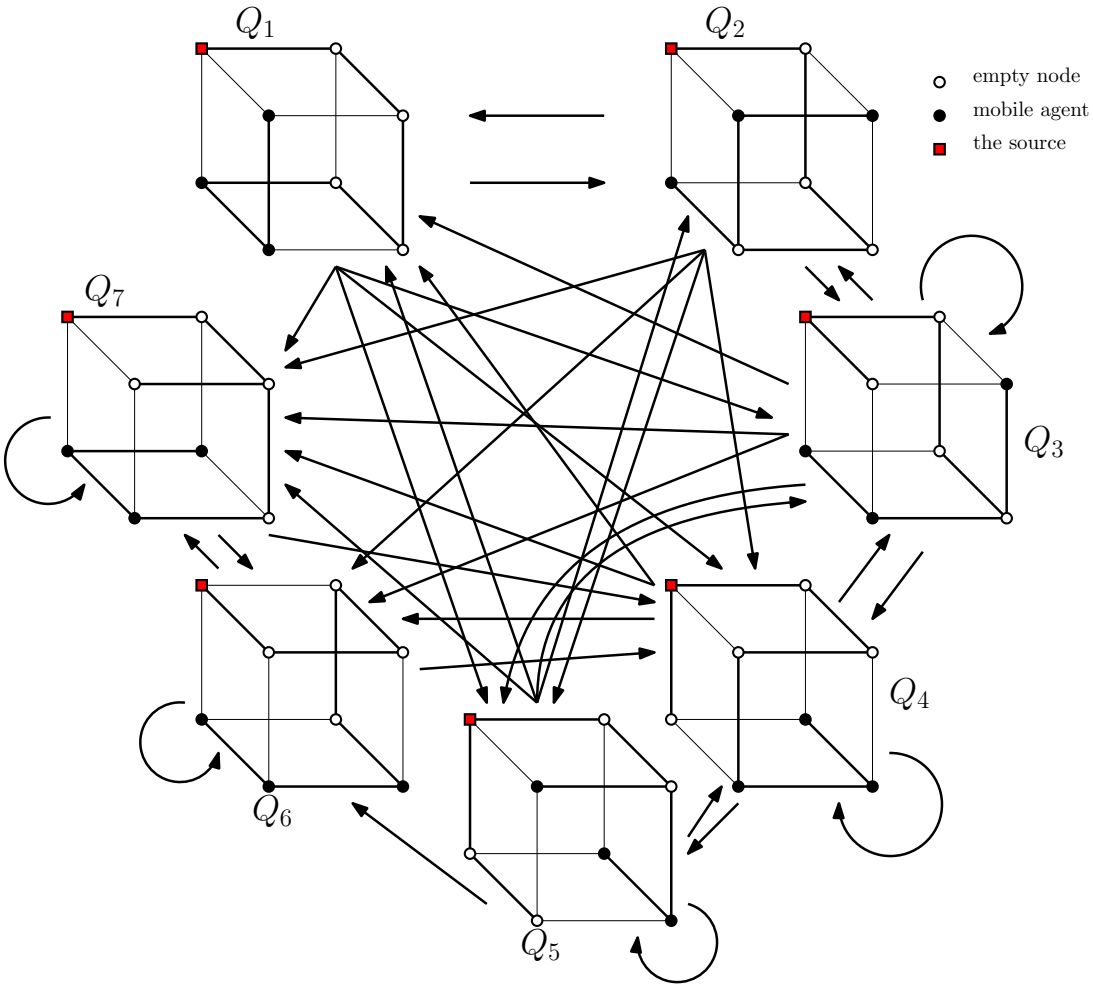

FIGURE 5.3: Bad configurations for $k = 3$ agents (denoted by black discs) and one source (denoted by red square) in a cube. In each case the adversary makes available only the bold edges. The arrows denote all possible transformations between configurations in one step of agent moves.

Q_3 $([100], [110], [101])$

Q_4 $([101], [011], [111])$

Q_5 $([100], [110], [111])$

Q_6 $([100], [011], [111])$

Q_7 $([100], [101], [011])$

Note that starting from any other initial configurations with 3 agents permits a solution to broadcast. However, when the 3 ignorant agents start in any configuration isomorphic to configurations in QF, then the configuration in the next round can only be another configuration in QF. All of the available transitions are listed bellow.

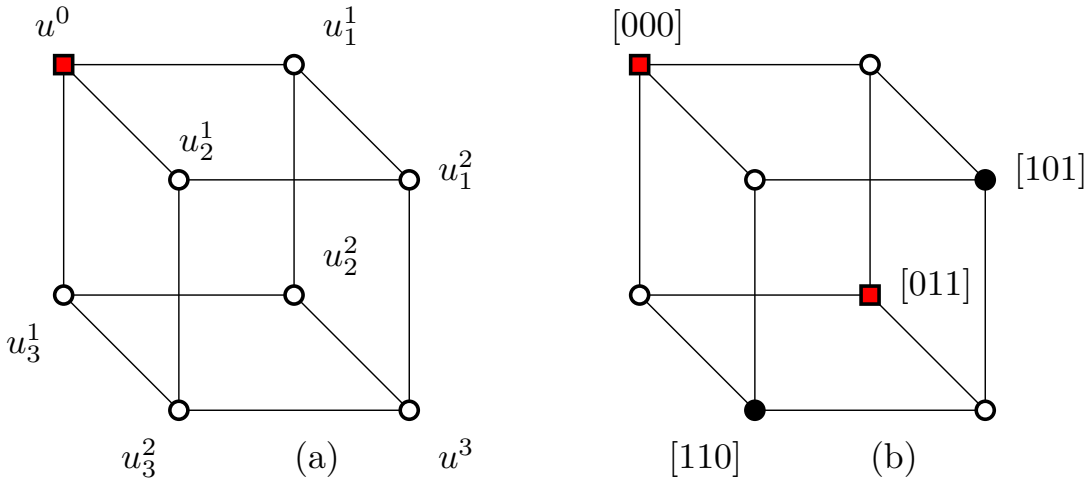

FIGURE 5.4: (a) The cube with a single source (denoted by a square) in the proof of Theorem 5.5. (b) The cube with two sources at distance 2; the remaining agents must occupy the two black vertices.

- $Q_1 \to \{Q_2, Q_3, Q_4, Q_5, Q_7\}$
- $Q_2 \to \{Q_1, Q_3, Q_4, Q_5, Q_6, Q_7\}$
- $Q_3 \to \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7\}$
- $Q_4 \to \{Q_1, Q_3, Q_4, Q_5, Q_6, Q_7\}$
- $Q_5 \to \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7\}$
- $Q_6 \to \{Q_4, Q_6, Q_7\}$
- $Q_7 \to \{Q_4, Q_6, Q_7\}$

We will demonstrate the transitions from Q_3. We get Q_1 if two agents and the source move to the same face of the cube. We get Q_2 if the source agent moves to the same face as the two diagonally positioned agents. We get Q_3 if the source agent and one agent from the bottom face change face and move toward the right as depicted. We get Q_4 if the two agents on the bottom face move. We get Q_5 if the source agent moves to the right face. We get Q_6 if the agent from the top face moves to the bottom face. And finally, we get Q_7 if the source moves above the agent on the front and the agent on the top face moves to the bottom.

This implies that no ignorant agent can meet the source after any number of rounds, and thus it is not possible to solve the Broadcast problem. □

Proof. (**Upper Bound**) We show that $k = 4$ agents can solve broadcast by providing an algorithm. We first show that one of the four agents can meet the source agent. We denote the vertices of the cube as follows: $u^0 = [000]$ is the vertex containing the source, u_1^1, u_2^1, u_3^1 are the three vertices at distance one from source, u_i^1 means a generic vertex at distance 1 from the source, u_1^2, u_2^2, u_3^2 are the three vertices at distance 2 from the source, and $u^3 = [111]$ is the only vertex at distance 3 (see Figure 5.4(a)). We now consider all possible initial configurations with agents placed on distinct vertices; Note that, each such configuration has at least three empty vertices. We denote such a configuration as $[x, y, z, l]$ showing the positions of the 4 ignorant agents at vertices x, y, z, l (with $*$ denoting any vertex other than the source).

$C1$ Configuration $[u_1^1, u_2^1, u_3^1, *]$: At least one of the links (u^0, u_1^1), (u^0, u_2^1), or (u^0, u_3^1) must be active, otherwise vertex u^0 would be disconnected. Hence at least one agent can meet the source within the next time unit.

$C2$ Configuration $[u_1^2, u_2^2, u_3^2, *]$: At least one of the paths of distance two between the source vertex u^0 and one of the vertices u_1^2, u_2^2, or u_3^2 must be available, otherwise vertex u^0 would be disconnected from vertices u_1^2, u_2^2 and u_3^2. Hence, within the next step the agent at a distance two from the source, and the source agent move to the middle vertex of the path and meet.

$C3$ Configuration $[u_1^1, u_2^1, u^3, *]$: If at least one of the links (u^0, u_1^1) or (u^0, u_2^1) are active, then at least one agent can meet the source within the next time unit. Otherwise, the link (u^0, u_3^1) must be active (to ensure connectedness), so in that case, the source agent moves to vertex u_3^1. The configuration we obtain is isomorphic to the configuration $[C2]$ above, so we are done.

$C4$ Configuration $[u_1^1, u_2^1, u_1^2, u_2^2]$: Assume that there are no paths of length 1 or 2 from source to any agent (otherwise we are done as explained above). In that case, any possible spanning tree must have the edges (u^0, u_3^1) and (u_3^1, u_3^2) and further at least one of the edges (u_2^1, u_3^2) or (u_1^2, u^3) or (u_2^2, u^3). In the first case, one agent moves to u_3^2 and we obtain the configuration $[C2]$. In the other two cases, one agent moves to vertex u^3, and thus we obtain the configuration $[C3]$. So, we are done in all cases.

$C5$ Configuration $[u_i^1, u_1^2, u_2^2, u^3]$: In G_0, if there are no length-2 paths from source to any

agent, then any path from u^0 must go through u_3^2; If vertex u_3^2 has an available edge to some agent, this agent will move to u_3^2 and we would obtain the configuration [$C2$]. Otherwise u_3^2 has an available edge to some empty vertex $u_j^1 \neq u_i^1$ which is connected to some vertex occupied by an agent. Thus, this agent moves to u_j^1, and we obtain the configuration [$C3$].

We have shown one agent can reach the source within a constant number of steps and obtain the message. Now, there are two source agents and three ignorant agents. The two source agents can place themselves at distance two in G (this is always possible in at most 2 steps). Assume without loss of generality, that the two sources are at vertices $[000]$ and $[011]$ as in Figure 5.4(b). There are exactly two vertices ($[110]$ and $[101]$) that are at distance 2 from both the sources. If the ignorant agents occupy these two vertices, then in any spanning tree chosen by the adversary, at least one of the ignorant agents would be at distance two from a source agent. And in fact, it is easy to see that either the three ignorant agents occupy all three adjacent vertices of one of the two sources (which means that in the next step at least one more agent will meet a source), or three ignorant agents occupy two adjacent vertices of each source. In the last case (due to connectedness) at least two of the ignorant agents will occupy the two vertices ($[110]$ and $[101]$) in the next step. Finally, if only one of the ignorant agents occupies one of the vertices ($[110]$ and $[101]$), then in the next step this agent can move from that vertex and therefore we obtain the previous configuration (i.e., where three ignorant agents occupy two adjacent vertices of each source). Thus eventually at least two of the ignorant agents will occupy the two vertices ($[110]$ and $[101]$) and in the next round, at least one more ignorant agent will meet a source and therefore we will have 3 source agents.

Note that the case of 3 sources and 2 ignorant agents is analogous to the case of 2 sources and 3 ignorant agents, while the case of one ignorant agent and four sources, is analogous to the initial situation with one source and four ignorant agents. So, using the same strategies as above eventually all agents will obtain the message and broadcasting is solved. □

For hypercubes of higher dimensions $d \geq 4$, we do not have any general strategy for solving the problem as the adversary has too many possible ways of choosing the available

subgraph. However, the lower bound of $k = n/2 - 1$ agents from Theorem 5.4 still holds and we have the upper bound of $k = n - 2$ (from Observation 1).

5.6 Discussion

In this chapter, we studied the problem of broadcast for mobile agents moving in constantly connected dynamic networks. The main objective is to understand how many agents are necessary and sufficient to allow broadcast to be solved in various topologies. It turns out that for sparse topologies such as rings and cactus graphs, the number of agents needed for solving the broadcast problem can be independent of the network size n, while for denser graphs including grids, hypercubes, as well as the complete graph, $\Theta(n)$ agents are needed. This preliminary investigation on broadcast in dynamic graphs opens many new research directions. For both grids and hypercubes, we have large gaps between the lower bounds of $(n - 2\sqrt{n})$ and $(n/2 - 1)$ respectively, and the upper bound of $(n - 2)$. It seems that solving the problem in grids requires more agents than in hypercubes, since grid networks contain more redundant edges. However, the lower bound on hypercubes shows that the number of agents needed can sometimes be much more than the number of redundant edges in the network. This is in contrast to the *cops and robbers* problem where the number of cops needed is roughly equal to the number of redundant edges in the underlying graph [5]. In the future, we would like to study the differences between various problems in this model and try to adapt techniques used for broadcast, to solve other problems in dynamic networks. Moreover it would be nice to classify various problems according to the resources needed for solving them under the adversarial model studied in this paper. Another possible direction of research would be to replace the strong assumption of global visibility with some weaker assumptions about the agent's capabilities that still suffices to solve broadcast in this model.

6 Discussion

To sum up the results presented here we give a summary of each chapter respectively. In Chapter 3 we studied the problem of gathering under two visibility models, specifically, the global and local visibility models. In the global visibility model we have shown that two agents cannot achieve rendezvous in the presence of a malicious agent. Moreover, we present an algorithm for three or more agents that can achieve gathering. In the local visibility the impossibility result still holds, and we provide two algorithms that use fundamentally different techniques. The algorithms are for four and three or more agents respectively. All results except the algorithm for three or more agents in the local visibility model have been published in [34]. In Chapter 4 we have shown that nine agents are not enough to decontaminate an oriented and labeled ring, while on the case of a synchronous ring without sense of direction eleven agents are not enough to achieve the decontamination of the ring. We present and give proof of correctness of an algorithm with twelve agents that are able to decontaminate the synchronous unoriented ring in $O(n^2 + L)$ time units. In the oriented case we can achieve decontamination of the ring with ten agents in $O(n)$ time units. All results from this chapter have been published in [56]. Finally, in Chapter 5 we provide lower and upper bounds on the number of ignorant agents needed in order for a source agent to be able to broadcast some message for different network topologies from those that are sparser to those that are denser. Essentially, in sparse network topologies such as rinds and cactus graphs, the number of agents needed for solving the broadcast problem is independent of the network size n, while for denser graphs including grids, hypercubes, as well as the complete graph, $\Theta(n)$ agents are needed. Specifically, for the ring network, if the size is at least 5 then broadcast can be solved with at least 2 ignorant agents in $O(n)$ steps. If the size is less than 5 then broadcast can be solved with at least 1 ignorant agent in at most 2 steps. In the case of the cactus network if there are no cycles of size at least 5 (large cycles), and there are

only cycles of size less than 5 (small cycles), then broadcast is solvable, in $O(n)$ steps, with a number of ignorant agents at least the number of small cycles in the network. If there are also large cycles then in $O(n)$ steps broadcast is solvable with a number of ignorant agents which is at least the sum of the number of small and large cycles plus one. In the case of a $2 \times L$ grid network, we need at least L ignorant agents, in the case of a clique at least $n - 2$ agents are needed to solve broadcast in $O(n)$ steps, and in the case of the hyper cube network we need at least $n/2$ agents if the dimension is 3. In general for any dimension larger than 2 we have a lower bound on the number of agents of $n/2 - 1$.

Problem	Context	Agents	Complexity
Gathering	global vis.	$k < 3$	impossibility
	local vis.	$k \geq 3$	$O(n)$
Black virus decontamination	oriented ring	$k \geq 10$	$O(n)$
	unoriented ring	any k anonymous	impossibility
	unoriented ring	$k \geq 12$	$O(n^2 + L)$
	with advice $O(\log L)$	$k \geq 6$	$O(n)$
Broadcasting	ring $n \geq 5$	$k \geq 2$	$O(n)$
	ring $n < 5$	$k \geq 1$	2
	cactus $c2 = 0$	$k \geq c1$	$O(n)$
	cactus $c2 > 0$	$k \geq c1 + c2 + 1$	$O(n)$
	grid $2 \times L$	$k \geq L$	$O(n)$
	grid $W \times L$	$k < (L-1)(W-1)$	impossibility
	clique	$k \geq n - 2$	$O(n)$
	hypercube H_d	$k < n/2 - 1$	impossibility
	hypercube H_3	$k \geq 4$	$O(1)$

TABLE 6.1: Final results.

In conclusion we wanted to study problems in a distributed environment keeping all the advantages of a distributed context. For example, a road network where each driver has the role of a mobile agent is by nature distributed. Moreover, each driver, depending on the context, competes or cooperates with the other drivers in order to reach her destination faster. In such an environment having a central authority that decides who moves to where can be rather costly, the communications will be flooded and cramped, and we might not want that central authority due to the single point of failure, or because distributed decisions look to be more just.

The second point in our research was that if we fix the distributed environment with all the advantages and disadvantages that come with it, how much fault tolerant can such an environment be. How far can we go in terms of hostile entities and still be able to solve problems in such aggressive and hostile environments. In the problems of gathering and black virus decontamination, we followed the strategy of keeping the hostile entity the same and changing the model a little in order to find how many resources are needed that can be used to solve the problem. We found that each particular problem can actually be solved with very little resources at hand. In the broadcast problem we followed a different approach, we kept the model the same in all cases and changed only the network topology. In the ring topology we found that the resources needed to solve the problem is constant and do not depend on the size of the network. In the cactus the resources depend on the number of structural elements, on the general grid it looks like the resources needed are very close to the size of the grid, as well as in the case of a complete graph. In the general case of a hypercube network the resources are very dependent on the size and thus on the dimension, but we wanted to see if and how we could solve the problem and we studied the case of the cube specifically. Finally, we can actually solve most of those problems having only very little resources even if there is a hostile behaviour. The next steps on our research would be to consider more general and arbitrary topologies or make the environment even more hostile, in order to study where is the point from which we cannot solve those problems.

Bibliography

[1] BD Acharya and MK Gill. On the index of gracefulness of a graph and the gracefulness of two-dimensional square lattice graphs. *Indian J. Math*, 23(81-94):14, 1981.

[2] N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM J. on Computing*, 36(1):56–82, 2006.

[3] Modhawi Alotaibi. Black virus disinfection in chordal rings. Master's thesis, Université d'Ottawa/University of Ottawa, 2014.

[4] Balasingham Balamohan, Paola Flocchini, Ali Miri, and Nicola Santoro. Time optimal algorithms for black hole search in rings. *Discrete Mathematics, Algorithms and Applications*, 3(04):457–471, 2011.

[5] Stefan Balev, Juan Luis Jiménez Laredo, Ioannis Lamprou, Yoann Pigné, and Eric Sanlaville. Cops and robbers on dynamic graphs: Offline and online case. In *Proc. Structural Information and Communication Complexity - 27th International Colloquium, SIROCCO 2020*, volume 12156 of *Lecture Notes in Computer Science*, pages 203–219. Springer, 2020.

[6] E. Bampas, N. Leonardos, E. Markou, A. Pagourtzis, and M. Petrolia. Improved periodic data retrieval in asynchronous rings with a faulty host. *Theoretical Computer Science*, 608:231–254, 2015.

[7] Eduardo Mesa Barrameda, Shantanu Das, and Nicola Santoro. Deployment of asynchronous robotic sensors in unknown orthogonal environments. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 125–140. Springer, 2008.

[8] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *Proc. of 14th ACM Symp. on Parallel Algorithms and Architectures*,

pages 200–209, 2002.

[9] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Theory of Computing Systems*, 44(3):143–162, 2007.

[10] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209. ACM, 2002.

[11] Lali Barriere, Paola Flocchini, Eduardo Mesa-Barrameda, and Nicola Santoro. Uniform scattering of autonomous mobile robots in a grid. *International Journal of Foundations of Computer Science*, 22(03):679–697, 2011.

[12] Lélia Blin, Pierre Fraigniaud, Nicolas Nisse, and Sandrine Vial. Distributed chasing of network intruders. *Theoretical Computer Science*, 399(1-2):12–37, 2008.

[13] Z. Bouzid, S. Das, and S. Tixeuil. Gathering of mobile robots tolerating multiple crash faults. In *ICDCS 2013*, pages 337–346, 2013.

[14] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Mobile-agent coordination models for internet applications. *Computer*, 33(2):82–89, 2000.

[15] J Cai, P Flocchini, and Nicola Santoro. Decontamination of an arbitrary network from multiple black viruses. In *32nd International Conference on Computers and Their Applications, (CATA)*, pages 231–237, 2017.

[16] Jie Cai, Paola Flocchini, and Nicola Santoro. Decontaminating a network from a black virus. *International Journal of Networking and Computing*, 4(1):151–173, 2014.

[17] Jie Cai, Paola Flocchini, and Nicola Santoro. Black virus decontamination in arbitrary networks. In *New Contributions in Information Systems and Technologies*, pages 991–1000. Springer, 2015.

[18] Jie Cai, Paola Flocchini, and Nicola Santoro. Distributed black virus decontamination and rooted acyclic orientations. In *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*, pages 1681–1688. IEEE, 2015.

[19] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[20] J. Chalopin, Y. Dieudonne, A. Labourel, and A. Pelc. Rendezvous in networks in spite of delay faults. *Distributed Computing*, 29:187–205, 2016.

[21] J. Chalopin, E. Godard, Y. Métivier, and R. Ossamy. Mobile agent algorithms versus message passing algorithms. In *Proc. of Principles of Distributed Systems*, LNCS 4305, pages 187–201, 2006.

[22] M. Chrobak, L. Gasieniec, and W. Rytter. Fast broadcasting and gossiping in radio networks. *J. Algorithms*, 43(2):177–189, 2002.

[23] H. Chuangpishit, J. Czyzowicz, E. Kranakis, and D. Krizanc. Rendezvous on a line by location-aware robots despite the presence of byzantine faults. In *ALGOSENSORS 2017*, pages 70–83, 2017.

[24] Colin Cooper, Ralf Klasing, and Tomasz Radzik. Locating and repairing faults in a network with mobile agents. *Theoretical Computer Science*, 411(14-15):1638–1647, 2010.

[25] Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *International Symposium on Distributed Computing*, pages 148–162. Springer, 2010.

[26] J. Czyzowicz, K. Diks, J. Moussi, and W. Rytter. Energy-optimal broadcast and exploration in a tree using mobile agents. *Theoretical Computer Science*, 795:362–374, 2019.

[27] J. Czyzowicz, R. Killick, E. Kranakis, D. Krizanc, and O. Morale-Ponce. Gathering in the plane of location-aware robots in the presence of spies. In *SIROCCO 2018*, 2018.

[28] Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability and Computing*, 16(4):595–619, 2007.

[29] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro. Map construction of unknown graphs by multiple agents. Theoretical Computer Science, 385(1-3):34–48, 2007.

[30] S. Das, P. Flocchini, A. Nayak, and N. Santoro. Effective elections for anonymous mobile agents. In *Proc. of 17th Int. Symp. on Algorithms and Computation*, pages 732–743, 2006.

[31] S. Das, P. Flocchini, N. Santoro, and M. Yamashita. Fault-tolerant simulation of message-passing algorithms by mobile agents. In *Proc. of 14th Int. Colloquium on Structural Information and Communication Complexity*, LNCS 4474, pages 289–303, 2007.

[32] S. Das, R. Focardi, F.L. Luccio, E. Markou, D. Moro, and M. Squarcina. Gathering of robots in a ring with mobile faults. In *17th Italian Conference on Theoretical Computer Science (ICTCS 2016), Lecce, Italy*, pages 122–135. CEUR, Vol 1720, September 7-9, 2016.

[33] S Das, R. Focardi, F.L. Luccio, E. Markou, and M. Squarcina. Gathering of robots in a ring with mobile faults. *Theoretical Computer Science*, 2018. In press https://doi.org/10.1016/j.tcs.2018.05.002.

[34] S. Das, N. Giachoudis, F. L. Luccio, and E. Markou. Gathering of robots in a grid with mobile faults. In *45th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2019)*, volume 11376 of *LNCS*, pages 164–178. Springer, 2019.

[35] S. Das, F.L. Luccio, and E. Markou. Mobile agents rendezvous in spite of a malicious agent. In *ALGOSENSORS 2015*, LNCS 9536, pages 211–224, 2015.

[36] Shantanu Das, Dariusz Dereniowski, and Christina Karousatou. Collaborative exploration of trees by energy-constrained mobile robots. *Theory of Computing Systems*, 62(5):1223–1240, 2018.

[37] Shantanu Das, Nikos Giachoudis, Flaminia L. Luccio, and Euripides Markou. Broadcasting with Mobile Agents in Dynamic Networks. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, volume 184 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[38] Dariusz Dereniowski. Connected searching of weighted trees. *Theoretical Computer Science*, 412(41):5700–5713, 2011.

[39] Y. Dieudonne, A. Pelc, and D. Peleg. Gathering despite mischief. *ACM Transactions on Algorithms*, 11(1):1, 2014.

[40] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Multiple agents rendezvous in a ring in spite of a black hole. In *OPODIS 2003*, pages 34–46, 2003.

[41] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48(1):67–90, 2007.

[42] Stefan Dobrev, Paola Flocchini, Rastislav Královič, and Nicola Santoro. Exploring an unknown dangerous graph using tokens. *Theoretical Computer Science*, 472:28–45, 2013.

[43] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. *Distributed Computing*, 19(1):1–99999, 2006.

[44] A. Ferreira. Building a reference combinatorial model for manets. *IEEE Network*, 18(5):24–29, 2004.

[45] P. Flocchini, B. Mans, and N. Santoro. Sense of direction: Definitions, properties, and classes. *Networks*, 32(3):165–180, 1998.

[46] P. Flocchini, B. Mans, and N. Santoro. Sense of direction in distributed computing. *Theoretical Computer Science*, 291:29–53, 2003.

[47] P. Flocchini and N. Santoro. Distributed security algorithms for mobile agents. In J. Cao and S.K. Das, editors, *Mobile Agents in Networking and Distributed Computing*, chapter 3, pages 41–70. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2012.

[48] Paola Flocchini, Miao Jun Huang, and Flaminia L Luccio. Decontamination of hypercubes by mobile agents. *Networks: An International Journal*, 52(3):167–178, 2008.

[49] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile entities. *Current Research in Moving and Computing*, 11340, 2019.

[50] Paola Flocchini, Alessandro Roncato, and Nicola Santoro. Backward consistency and sense of direction in advanced distributed systems. *SIAM Journal on Computing*, 32(2):281–306, 2003.

[51] Fedor V Fomin, Dimitrios M Thilikos, and Ioan Todinca. Connected graph searching in outerplanar graphs. *Electronic Notes in Discrete Mathematics*, 22(213-216):7th, 2005.

[52] P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48:166–177, 2006.

[53] L. Gasieniec. *Deterministic Broadcasting in Radio Networks*, pages 233–235. Springer US, Boston, MA, 2008.

[54] L. Gasieniec and A. Pelc. Adaptive broadcasting with faulty nodes. *Parallel Computing*, 22(6):903–912, 1996.

[55] L. Gasieniec and A. Pelc. Broadcasting with linearly bounded transmission faults. *Discrete Applied Mathematics*, 83(1-3):121–133, 1998.

[56] Nikos Giachoudis, Maria Kokkou, and Euripides Markou. Black virus decontamination of synchronous ring networks by initially scattered mobile agents. In *International Colloquium on Structural Information and Communication Complexity SIROCCO 2020*, pages 220–236. Springer, 2020.

[57] F. Harary and G. Gupta. Dynamic graph models. Mathematical and Computer Modelling, 25(7):79–88, 1997.

[58] Frank Harary, John P. Hayes, and Horng-Jyh Wu. A survey of the theory of hypercube graphs. *Computers & Mathematics with Applications*, 15(4):277–289, 1988.

[59] Tien-Ruey Hsiang, Esther M Arkin, Michael A Bender, Sándor P Fekete, and Joseph SB Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Algorithmic Foundations of Robotics V*, pages 77–93. Springer, 2004.

[60] D. Ilcinkas and A.M. Wade. Exploration of the t-interval-connected dynamic graphs: the case of the ring. *Theory of Computing Systems*, 62(5):1144–1160, 2018.

[61] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary graphs. *TCS*, 384(2-3):201–221, 2007.

[62] Maria Kokkou. Distributed computing - fault tolerant distributed algorithms. Bachelor's Thesis, University of Thessaly, 2019.

[63] E. Kranakis, D. Krizanc, and E. Markou. *The Mobile Agent Rendezvous Problem in the Ring*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2010.

[64] F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd Symposium on Theory of Computing (STOC)*, pages 513–522, 2010.

[65] J. Lin, A. S. Morse, and B. D. O. Anderson. The Multi-Agent Rendezvous Problem. An Extended Summary. *Cooperative Control*, 309:451–454, 2004.

[66] Yichao Lin. Decontamination from black viruses using parallel strategies. Master's thesis, Université d'Ottawa/University of Ottawa, 2018.

[67] Flaminia L. Luccio. Contiguous search problem in sierpinski graphs. *Theory Comput. Syst.*, 44(2):186–204, 2009.

[68] N. Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.

[69] Euripides Markou, Evangelos Kranakis, Aris Pagourtzis, and Danny Krizanc. *Αλγοριθμική θεωρία κατανεμημένων υπολογισμών. [ηλεκτρ. βιβλ.]*. Αθήνα:Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών, 2015.

[70] Euripides Markou and Wei Shi. Dangerous graphs. In *Distributed Computing by Mobile Entities*, pages 455–515. Springer, 2019.

[71] Nicolas Nisse. Connected graph searching in chordal graphs. *Discrete Applied Mathematics*, 157(12):2603–2610, 2009.

[72] Nicola Santoro. *Design and Analysis of Distributed Algorithms*. Wiley-Blackwell, 2006.

[73] CL. E. Shannon. Presentation of a maze-solving machine. In *Proc. of 8th Conf. of the Josiah Macy Jr. Found. (Cybernetics)*, pages 173–180, 1951.

[74] Richard J Trudeau. *Introduction to graph theory*. Courier Corporation, 2013.

[75] Eric W Weisstein. Grid graph. *https://mathworld. wolfram. com/*, 2001.

[76] Eric W Weisstein. Minimum vertex cut. *https://mathworld. wolfram. com/*, 2001.

[77] Eric W Weisstein. Vertex cut. *https://mathworld. wolfram. com/*, 2001.

[78] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[79] Yukiko Yamauchi, Tomoko Izumi, and Sayaka Kamei. Mobile agent rendezvous on a probabilistic edge evolving ring. In *ICNC*, pages 103–112, 2012.