A Rupestrian Algorithm*

Giuseppe A. Di Luna[1], Paola Flocchini[2], Giuseppe Prencipe[3], Nicola Santoro[4], and Giovanni Viglietta[5]

1 **School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada**
gdiluna@uottawa.ca
2 **School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada**
flocchin@site.uottawa.ca
3 **Dipartimento di Informatica, Unversità di Pisa, Pisa, Italy**
giuseppe.prencipe@unipi.it
4 **School of Computer Science, Carleton University, Ottawa, Canada**
santoro@scs.carleton.ca
5 **School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada**
viglietta@gmail.com

Abstract

Deciphering recently discovered cave paintings by the Astracinca, an egalitarian leaderless society flourishing in the 3rd millennium BCE, we present and analyze their shamanic ritual for forming new colonies. This ritual can actually be used by systems of anonymous mobile finite-state computational entities located and operating in a grid to solve the *line recovery problem*, a task that has both self-assembly and flocking requirements. The protocol is totally *decentralized*, fully *concurrent*, provably *correct*, and time *optimal*.

1998 ACM Subject Classification C.2.4 Distributed Systems, F.1.2 Modes of Computation, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases mobile finite-state machines, self-healing distributed algorithms

Digital Object Identifier 10.4230/LIPIcs.FUN.2016.14

1 Introduction

1.1 Anthropological Discovery

Recently, an archaeological expedition in Budelli has discovered cave paintings and pottery from the 3rd millennium BCE. The artifacts have been attributed to the Astracinca semi-nomadic tribes, whose villages were located in a wide area of northern Sardinia. The Astracinca civilization, undeservedly little known, stood out for the highly sophisticated social organization and the advances in ethno-botanical techniques.

The abundance of resources, due to efficient agricultural techniques, and the high fertility, due to a joyously promiscuous social organization, cyclically brought each village to address the problem of overpopulation. As in many other civilizations, that problem was solved by

* This research has been supported in part by: the Natural Sciences and Engineering Research Council (Canada) under the Discovery Grant program, Prof. Flocchini's University Research Chair, and project PRA_2016_64 "Through the fog" funded by the University of Pisa.

8th International Conference on Fun with Algorithms (FUN 2016).
Editors: Erik D. Demaine and Fabrizio Grandoni; Article No. 14; pp. 14:1–14:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

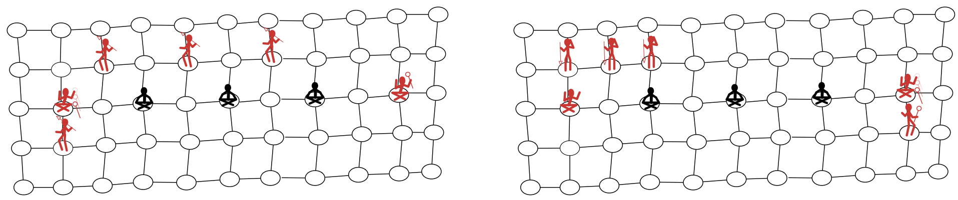


Figure 1 Reproduction of rupestrian paintings discovered in Budelli Island.

selecting a group of settlers, or two of equal size, with the task of creating new settlements, thus solving the overpopulation problem.

Unlike other civilizations, the Astracinca were an egalitarian leaderless society: upon reaching puberty, all members of the tribe were socially equal and decisions were reached by consensus. All agreed-upon social procedures guiding major events were carried out without any member being pre-assigned a special role. Such social procedures have been shrouded in the deepest mystery, and their complexity had been only inferred by cultural anthropologists, because no archaeological finding describing the procedures was available so far.

The found cave paintings (two of which are shown in Figure 1) are remarkable because they contain the description of the procedure used by the Astracinca to create a new settlement, an event of major importance in the life of the tribe; hence the great excitement in deciphering the procedure, and the interest in its intricacies, unparalleled in history.

After a long study of the paintings and pottery found in the cave, the procedure has been finally deciphered; it is indeed composed of a set of intricate rules, defining a shamanistic ritual WonnaGo to be performed by the adult population of the village. Before providing the details, let us give an overview of the discovered process.

The ritual WonnaGo takes place in a large clearing, away from the village, on which a regular grid is engraved, each node of which is large enough to host a person. Before the start of the ritual, a large number of bowls filled with liquid are placed just outside the grid. Each villager picks up a bowl and then sits in an empty node, so to form a line.

Each bowl is filled with liquid from one of two preparations, indistinguishable from the outside; the proportion between the bowls filled by each preparation is unspecified. By analyzing the residues found in the pottery, we know the composition of the two potions. The first potion, based on poppy milk, induces in drinkers a deep comatose state, making movements and communication impossible. Those who consume this substance are catatonically locked in their starting position for the duration of the rite. Upon exiting their comatose state, they continue their life in the old village. The second potion, made from Amanita Muscaria and Cannabis Sativa, selects the *settlers*. The shamanic use of Amanita Muscaria to induce hallucinatory and dissociative states is well known [18].

Dissociative substances may cause a drastic decrease in vision and perceptions; this effect is known as Kalnienk vision [3]. Incredibly, the set of rules predict this eventuality: they only assume the ability to observe the nodes adjacent to the one occupied by a settler, and communication to be limited to the settlers in those locations. The knowledge of this phenomenon by Astracinca also explains the use of a grid with carefully studied proportions. The medical literature has a well-established knowledge of acute intoxication by phytocannabinoids, in particular the short-term deterioration of memory [14]. The rules include this possibility, since each settler needs to remember a limited amount of information that does not depend on the number of participants.

During WonnaGo, the inebriated settlers leave their comrades in a cathatonic state, and eventually form either a new line or two lines of equal length. All the settlers in the same line eventually become in agreement on the same direction. Once this happens, the ritual ends and the migration starts.

1.2 Technological Interpretations and Contributions

The social structure of the Astracinca civilization and the rules they use in their rituals are surprisingly modern. Indeed, not only do they have impressive similarities with alternative visions of how human systems could function (e.g., [16]), but also directly reflect a variety of current artificial systems ranging from robotic swarms to mobile sensor networks, from biologically inspired systems to mobile software agents. Indeed, systems of *mobile computational entities* that operate in a spatial universe, obeying the same set of rules in a totally decentralized way, with very limited (or no) local memory and communication capabilities, are almost ubiquitous. From an algorithmic point of view, such systems are being extensively and intensively studied, in particular within distributed computing. They include

the *metamorphic* robots (e.g., [5, 6, 20]), the amoeba-inspired *amebots* [8, 9], the finite-state *mobile robotic sensors* (e.g., [4, 13]), and the extensively studied *oblivious mobile robots* (e.g., [1, 7, 11, 12, 19]). In some of these systems the entities can move in a continuous spaces (e.g., $\mathbb{R}^2$); in others the movements occur in a discrete space (e.g., $\mathbb{Z}^2$). In particular, extensive investigations have been carried out when the computational entities are identical finite-state machines operating and moving in a (possibly incomplete) grid (e.g., [4, 8, 9, 10, 13, 15, 17]).

The model implied by the Astracinca ritual is precisely that of anonymous mobile finite-state machines located and operating in a grid in a fully synchronous system. Each entity is a finite-state machine, can move in the grid from node to node, and is able to communicate with the entities located at neighboring nodes; since the entities are finite-state, the amount of information exchanged in a communication is bounded by a constant. The entities are anonymous and behaviorally identical; that is they have no distinguished identifiers and they all execute the same algorithm. The entities do not rely on a common coordinate system: each entity fixes a local orientation of the grid, but different entities may have different orientations. How the communication between two neighboring entities is performed (e.g., accessing a shared variable, reading the other entity's state, wireless transmission, etc.) is irrelevant for our investigation. Analogously irrelevant is how an entity performs its movement (e.g., extending and contracting its body, using wheels, transported by a service robot, etc).

The problem addressed by the WonnaGo ritual has two aspects. It is first of all a problem of *self-assembly* and *self-repair* of the system: initially located on a line, upon the (possible) failure of some entities, the non-faulty ones must reform the line excluding any faulty element. Solving this problem requires formulating a set of rules (the algorithm) that will allow the entities to form the line within finite time, regardless of the initial distribution and number of faults and of the local orientations of the non-faulty entities. It is also a problem of *coordinated moving* or *flocking*: the non-faulty entities must move away (possibly forever) while maintaining the line formation.

Unfortunately both these tasks, as formulated, are actually *unsolvable*, even in fully synchronous systems. In fact, there are initial configurations where unbreakable symmetries make it impossible to form a single line. Similarly, even if the non-faulty entities are all on a single line (e.g., if there are no faulty entities), if their number is even, there are assignments of the local orientations that render the flocking of the line impossible. Both impossibilities are circumvented by requiring that either one or two lines of equal size be formed, and that each formed line migrates maintaining its line formation; we shall call this problem *line recovering*. The Astracinca must have been aware of those impossibilities; in fact, the WonnaGo ritual meets precisely this requirement.

In this paper we present the deciphered ritual and analyze its effectiveness. We prove that the set of deciphered rules always and correctly leads the non-comatose participants to form either a single line or two lines of equal size, and each line moves to find new locations. We also prove that the short ritual allows the process to be performed in optimal time.

Additionally, we provide a ritual simulator. The C source code, a pre-compiled binary for Windows systems, and the relative instructions can be found at `http://giovanniviglietta.com/files/rupestrian/Simulator.zip`.

Summarizing, we show that the Astracinca have developed a synchronous protocol for the *line recovery problem*; the protocol is totally *decentralized*, fully *concurrent*, provably *correct*, and *time optimal*.

2 System Model and Rite Purpose

We consider the space to be an infinite unoriented anonymous mesh $G(V, E)$, i.e., the nodes in V are all equal, edges are bidirectional and unlabeled, G is constituted by an infinite number of rows and columns. The tribe is a set of n persons in distinct positions in G. Each person p is modeled as a tuple (x, s, dir, pre, b), where $x \in V$ represents the person's *position*, $s \in S$ is a *state* (where S is a fixed finite set of possible states, with $S \supseteq \{sleeper, settler\}$), $dir, pre \in D = \{up, down, left, right, none\}$ represent two *directions* to which the person is pointing (the current direction dir and the previous location pre), and $b \in \{0, 1\}$ is a *bump flag*.

The purpose of the bump flag is to let a person know that they bumped into someone while trying to move to a location, as will be explained shortly.

Given a node $x \in V$, $N(x)$ is the set of its four neighbors. For explanation purposes, we use a global reference system, which allows us to give consistent labels in $D \setminus \{none\}$ to the edges from x to $N(x)$. This reference system is unknown to the persons. Each person p has their own reference system: when on node x, person p associates to each node in $N(x)$ a direction in $D \setminus \{none\}$. The *neighborhood* of p in x is an ordered list of elements $\{up, down, left, right\}$. An element is *empty* if on the corresponding node, with respect to the reference system of p, there is no person. Otherwise, if there is an agent $p' = (y, s', dir', pre', b)$ in that direction, then the element is $(s', T(p, p', dir'), T(p, p', pre'))$, where T is a function that translates the direction d' of p' in the reference system of p. Essentially, a person can see the states of persons on the neighboring nodes and their directions, but not their coordinates and their bump flags.

Time is divided in fixed size intervals (*rounds*), $r \in \mathbb{N}^+$. At each round r, every person with state $s \neq sleeper$ is activated. Upon activation, a person performs some operations based on their *view* (i.e., their state, directions and neighborhood at the beginning of r). The operations to be executed are determined according to a set of rules, called *ritual*, which is the same for all persons. Given a person's view at the beginning of a round, the ritual specifies whether the person must move and where, and indicates the new state and the directions at the end of the round. A person in node x at round r may move to any $y \in N(x)$. If at round r several persons move towards the same empty node y, only one of them succeeds and will be at node y at the beginning of round $r + 1$. All other persons remain in their nodes and at the beginning of round $r + 1$, and will have the bump flags set.

Given a set of persons we say that they are on a *straight line* if they are all on the same row or column of G. We say that they are on a *compact straight line* if they are on a *straight line* and the subgraph induced by their positions is connected. We say that a set of persons is oriented in direction d if their directions dir once translated in the global reference system are all equal to d. Initially, at round $r = 1$, all persons are positioned on a compact straight line (the "initial line"), $f \geq 0$ of them have state *sleeper*, all the other $n - f \geq 5$ have state *settler* and directions set to *none*.

The *line recovery* problem is solved at round r^* if, for any round $r \geq r^*$, the initial *settlers* form a straight line and they are oriented in a certain direction, or they form two straight lines of equal size but with opposite orientations.

3 The WonnaGo Ritual

3.1 Overall Description

In the following, we will assume that persons can exchange fixed-size messages: this can be easily simulated in our model. Also, if not otherwise specified, the variable *dir* of a person p stores the movement's direction of p, that is, it stores the location where p intends to move; when no ambiguity arises, we will use the expression "direction of p" to indicate the content of *dir*. Similarly, the content of variable *pre* stores the location of p in the previous round; again, when no ambiguity arises, we will use the expression "previous location of p" to denote the content of this variable. We will say that p is *pointing at* a person p' if p and p' are neighbors, and the direction *dir* of p is toward the location occupied by p'. Finally, when a person p changes state from s, we will say that p *becomes* s.

The WonnaGo ritual is divided in several sub-rites: Exodus, Explorer Divination, Marker Creation, Chief Identification, The Chosen One, Opposite Sides, and Same Side. Let L_0 be the row where the initial line is placed, and L_1, L_{-1} the two rows adjacent to L_0.

The rite starts by checking whether there are no *sleepers*: in this case, all *settlers* are already in a compact line. This scenario is detected during the Exodus sub-rite, started (at the first round) by the settlers who occupy the extreme positions of the starting configuration, i.e., by the two persons having only one neighbor. These two extreme *settlers* send a special message inside the line: if the two messages meet, there are no *sleepers*; otherwise, the Exodus gets interrupted.

Should there be *sleepers*, the second sub-rite (the Explorer Divination) is performed, started by all the *settlers* who have a *sleeper* neighbor. In this sub-rite some *settlers* become *explorers* and move out of the line. The selection of the *explorers* is made in such a way that their movement does not create "gaps" of more than two consecutive empty positions anywhere in the original line (this property is crucial to detect the end of the line in subsequent sub-rites). Notice that it is possible that both the Exodus and the Explorer Divination sub-rite are started concurrently, in which case the Exodus process will die.

After stepping out of the initial line, the *explorers* start moving in a direction of their choice, and the Marker Creation sub-rite begins. The goal of this sub-rite is to place an *explorer* at each end of the original line so to mark it for subsequent rites. To achieve this, the *explorers* move along the chosen direction.

An *explorer* that sees the end of the line (i.e., three consecutive empty positions), moves immediately after it and becomes a *marker* with *scepter flag* set. After the *marker(s)* are created, the other explorers continue to move towards the end of the line: this is handled in the Chief Identification sub-rite.

The main goal of the Chief Identification sub-rite is to select at most two *explorers* as *chiefs*. In particular, when an *explorer* reaches a *marker* with scepter flag set, they become a *chief*, and the *marker* loses the scepter. Then, the *chief* inverts direction and tries to reach the other end of the line, i.e., the other *marker*. Due to concurrency, several scenarios can occur which make the *chief* understand whether it is unique or not. In case there are two *chiefs*, each of them will understand whether they are located in different lines (L_1, L_{-1}) or on the same line (L_1 or L_{-1}) with opposite directions. Depending on the situation, a new sub-rite begins (The Chosen One, Opposite Sides, or Same Side).

The Chosen One sub-rite handles the easiest of the three possible outcomes of the Chief Identification sub-rite: in this case one of the *chiefs* is the first to reach also the other *marker* with scepter flag set (or reaches an extreme of the line with no *marker* at all), and becomes the (unique) *chosen one*. During this sub-rite, the *chosen* moves through the line, collecting anyone who is not *sleeper*,

forming a procession that will complete the assigned task.

In the Opposite Sides sub-rite, the two *chiefs* realize (when they reach the *marker* at the opposite end of their respective line) to be located on two different lines (i.e., one on L_1 and the other on L_{-1}): in this case, each *chief* becomes a *collector*, and a two-phase process is started. In the first one, a *collector* goes to the other *marker*, moving every two rounds, and collecting all people encountered on the way as well as the *settlers* still on L_0. The second phase is a counting process, which is an attempt to establish which of the processions formed by the two *collectors* is the longest. If one of the two processions is longer, its *collector* is elected, performs a final loop of the line, collecting everybody, and eventually forms a unique straight line, thus completing the task. Otherwise, in the case the two processions have the same length, the two *chiefs* move along opposite directions, until two distinct straight lines are formed, thus completing the task.

The Same Side sub-rite occurs when a *chief* meets another *chief*: in this case, the two *chiefs* realize to be on the same line, say L_1, and that they are moving in opposite directions. As soon as the two *chiefs* meet, they become *opposers*, switch directions, and move along the new direction, collecting everybody they encounter along the way. When an *opposer* reaches a *marker*, they start the final *collecting* phase: they keep moving in the same direction (on L_{-1}) and collect the *settlers* still on L_0. Eventually, the two *opposers* meet on L_{-1}: at this point, messages are exchanged among the people within the processions led by the two *opposers*, in an attempt of electing one of the *opposers*. If this is possible, the *opposer* that gets elected starts moving until the procession forms a straight line, thus completing the task. Otherwise (i.e., it is not possible to elect a unique *opposer*), the two *opposers* move along opposite directions, until two straight lines of equal size are formed, thus completing the task.

In the following the sub-rites are detailed; the complete set of rules and the reproductions can be found at the end of the paper.

3.2 Sub-rites

Exodus. In the Exodus sub-rite the *settlers* detect if there are no *sleepers* in the system, and in that case they elect one or two *exodus.leaders*, who will lead the migration. This is done as follows: in the first round, a *settler* detects if they are at the extreme of the line, i.e., they have only one neighbor. If this is the case, the *settler* becomes a *marker* with scepter flag set. At the end of the first round the *marker* sends an "Exodus?" message to an active neighbor, if any exists. A *settler* receiving such a message propagates it to the next *settler*. If there are no *sleepers*, then the two "Exodus?" messages meet; at that point, depending on the parity of the number of *settlers*, either one or two lines of equal size are formed. Otherwise (i.e., there are *sleepers*) the Explorer Divination sub-rite is eventually performed.

Explorer Divination. The sub-rite Explorer Divination is used to bootstrap the other sub-rites, and it is executed if at least one *sleeper* is present. The purpose of the rite is to select at least three *explorers* among the *settlers*, who will move out of the line without creating empty "gaps" of more than two consecutive positions. This is done as follows. If a *settler* has a *sleeper* neighbor, then they become an *explorer* and they notify this decision to the neighboring *settler*, if any exists. Upon receiving such a notification, a *settler* becomes *settler.notified*. Any *settler.notified* who is not neighbor of another *settler.notified* becomes an *explorer* as well.

At this point (the fourth round) all *explorers* step out of the initial line.

Marker Creation. In this sub-rite the *explorers* move along the line until they find the end, which is detected by seeing three consecutive empty locations. The first *explorer* reaching the end of the line becomes a *marker* with scepter flag set and stays there. If two *explorers* try to become *marker* on the same extreme at the same time, only one is allowed to do so. Note that, when two *explorers* meet, they cannot pass through each other, and therefore they simply switch directions.

Depending on the initial configuration, either one or two *markers* are created by the end.

Chief Identification. The purpose of the Chief Identification sub-rite is to let at most two *explorers* become *chiefs*, and for a *chief* to determine if it is unique. An *explorer* who reaches a *marker* with scepter flag set receives the scepter flag from the *marker* and becomes a *chief*; should two *explorers* reach the same *marker* with scepter flag set at the same time, the *marker* will give the scepter to only one of them.

A newly elected *chief* now has to determine if they are the only one; to do so, they switch direction trying to reach the other extremity of the line. If the *chief* meets an *explorer* coming from opposite direction, it "virtually" continues its walk by switching roles with the explorer: the *explorer* becomes *chief* and switches direction, and the old *chief* becomes a *disciple* and stops. Similarly, if a *chief* and a *disciple* meet, they switch roles. If an *explorer* meets a *marker* without scepter flag or a *disciple*, it becomes *disciple* and stops.

There are three possible scenarios: (1) the *chief* reaches the other extremity, finding a *marker* with scepter flag or three empty locations; (2) the *chief* reaches the other extremity, finding a *marker* without scepter; (3) the *chief* meets another *chief*. Scenario (1) implies that the *chief* is unique; in this case the sub-rite The Chosen One starts. Scenario (2) implies that there are two *chiefs* who are moving on two different lines, adjacent to the initial one; in this case the sub-rite Opposite Sides is started. Scenario (3) means that there are two *chiefs* who are moving on the same line, adjacent to the initial one; in this case the sub-rite Same Side starts.

The Chosen One. The rules of the sub-rite The Chosen One are executed when a *chief* reaches a *marker* with scepter flag or detects that there is no *marker* on that side (three empty spots). When this happens, the *chief* becomes the *chosen*. The goal is for the *chosen* to collect everybody and eventually form a single line. To do so, the *chosen* reverses direction and moves, having everyone they meet follow them according to the Recruitment Procession rules described later. This movement is performed as follows: when the *chosen* meets a *disciple*, the *chosen* becomes a *follower* and the *disciple* becomes the *chosen*. When a *chosen* meets an *explorer*, a similar thing happens. In this process, a *settler* who sees a procession of *followers* will try to join the procession. Eventually, the *chosen* will reach the *marker*. When this happens, the *marker* becomes the *chosen*, and the *chosen* a *follower*. If there are no *disciples* around the *marker*, the *chosen* moves outside of the line and takes as direction the other endpoint of the line.

Opposite Sides. This sub-rite starts when a *chief* on L_1 (respectively, L_{-1}) reaches a *marker* m without the scepter flag: the chief understands that there is another *chief* moving in the same direction (clockwise or counter-clockwise) on L_{-1} (respectively, L_1). The *chief* becomes *collector*, switches direction, and moves toward the other *marker* m'; the *collector* moves every two rounds. During this swipe of L_1, the *collector* recruits all the encountered people, including the *settlers* still on L_0, thus forming a procession.

After at most $2n$ rounds, they reach the other *marker* m'. Let us assume for now that, during the swipe, they recruited at least one person. The *collector*, say x, and the closest recruited *follower*, say z, start now a phase to determine whether or not it is possible to form a unique procession. If not, two distinct processions of the same length will be formed. Specifically, at round r, z becomes *mover* and moves on L_1 towards *marker* m', eventually reaching it. Meanwhile, at round $r+2$, x becomes *collector.counting* and does not move (i.e., x remains close to *marker* m). After a finite number of rounds, a *marker* will have as neighbors both a *collector.counting* and a *mover*. When this occurs, the *marker* signals this event to both of them, say at round $r' > r+2$. At this time, the *collector.counting* and the *mover* simultaneously change state, becoming *probes*.

The *marker* also memorizes the line where the *collector.counting* lies; this information will be used to break possible symmetric scenarios.

Let us now focus on one of the two *markers*, say m: the two *probes* generated by m start moving towards m', one on L_1 and the other on L_{-1}, using the *probe move* protocol described below. In Section 4, we will prove that both *probes* reach m' at the same time if and only if the two processions formed by the two *collectors* have the same number of people; in fact, should one procession be smaller than the other, the *probe* traversing it will reach a *marker* before the other. In any case, each *marker* will know whether the two processions have the same length and, if not, which one is smaller.

The *probes* move according to the following protocol: each *probe* has a modulo-5 counter, initially set to 1. If the counter is greater than 0, the *probe* decrements it at each round. When the counter reaches 0, the *probe* moves to the next location. If the next location is empty, the *probe* moves and the counter remains 0. If, instead, the next location is occupied by a person p, then the *probe* "virtually" moves having p take the state of the *probe*, while the *probe* takes the state of p; in this case, p adds 1 to the counter (it adds another 1 to the counter if there is a *prefollower* pointing at p; refer to Section 3.3). If a *probe* reaches a *marker*, they first decrement the counter until 0, they then become a *follower* pointing towards the *marker*. There are two exceptions to this general rule: (E1) if at the same time two *probes* with counter 0 are neighbors of the same person p, they both take the state of p, p waits an appropriate number of rounds and signals to both neighbors (i.e., the old *probes*) to become *probe* pointing away from them with an appropriate set counter; (E2) two *probes* may move to the same empty location, with one of them bumping back. When this happens, both *probes* change direction but the one that did not bump sets the counter in such a way that they wait one round less; this is done to account for the fact that the bumping person did not move.

Eventually, a *probe* reaches a *marker*. If a *marker* detects that on the memorized side there is a new *probe* with counter 0, and on the other side there is no one or there is a *probe* with counter greater than 0, then the *marker* becomes a *consul* and points to the side opposite to the one they memorized. In this case a *consul* behaves as a *chosen* (see The Chosen One sub-rite), and does a loop around L_0 collecting all people (both the ones encountered on their way and the *settlers* still on L_0). Note that in this case the other *marker* does nothing, waiting to be collected. If at a given round a *marker* detects that on the memorized side there is a new *probe* with counter 0, and on the other side there is also a *probe* b with counter 0, then that the two processions have the same length (see Section 4). In this case, the *marker*, say m, becomes a *follower* and signals to b to become a *consul*; the *consul* will lead the procession away from L_0. Symmetrically, the same will happen on the side of the other *marker* m'.

At the beginning of this sub-rite, we assumed that the *collector* recruited at least one person. If this is not the case, when the *collector* reaches a *marker*, they become a *collector.counted* and move towards the other *marker*. If a *marker* sees a *collector.counted* and a *collector.counting*, they elect the *collector.counted*, appointing them a *consul*. Also in this case, a *consul* behaves as a *chosen*, and does a loop around L_0 collecting everyone encountered on the way (including the *settlers* still on L_0).

Same Side. In the Same Side case a *chief* meets another *chief*; let us assume, without loss of generality, they are both on L_1. When this happens both *chiefs* become *opposers*, they switch directions and move towards a *marker*. If an *opposer* moving towards a *marker*, say m, meets an *explorer* on its way, and their directions are opposite, the *opposer* continues moving collecting the *explorer* (i.e., the *opposer* becomes a *follower*, and the *explorer* becomes an *opposer* that still moves towards m). Eventually, the *opposer* reaches a *marker*: when this happens, the *opposer* moves to the spot occupied by the *marker* collecting the *marker* (i.e., the *opposer* becomes a *follower*, the *marker* becomes *opposer* and sets the follow-me flag, continuing to move in the same direction of the old *opposer*). From the next round on, the *opposer* (who is now on L_{-1}) starts collecting all encountered people (the ones met on L_{-1}, as well as the *settlers* still on L_0), thus forming their own procession.

Eventually, one of the two following scenarios can occur: (1) the two *opposers* are at distance 1 and between them there is a *disciple* (see Chief Identification sub-rite). In this case, both *opposers* become *followers* and the *disciple* becomes *opposer.winner*. (2) The two *opposers* meet on L_{-1}. When this occurs, they first wait two rounds (giving enough time to their immediate *followers* to reach them). Then, they both change state to *opposer*.waiting. At this point, each *opposer*.waiting starts a *straight line check* phase, by sending a *"straight line query"* to their procession (see Section 3.3) to figure out if it is possible to select a unique winner to lead the procession. Three cases can occur. (a) The two processions have different length: one of the two *opposer.waiting* is elected, becoming an *opposer.winner*, while the other *opposer* becomes a *follower*. (b) The two processions have equal length, and one procession is not already forming a straight line: the *opposer.waiting* who leads this procession becomes a *follower* and the other becomes *opposer.winner*. (c) The processions have equal length and they are both forming a straight line: in this case two *opposer.winners* are elected.

In cases (1), (2).a and (2).b, the only *opposer.winner* will lead the final migration; in case (2).c, the two *opposer.winners* set the flag tail, reverse direction, and each of them will lead the migration of their own procession.

3.3 Sub-Phases

The following two processes, *Recruitment Procession*, and *Straight Line Query*, are required in some of the rites described in the previous section to create and maintain a procession, and to compare two created processions, respectively.

Recruitment Procession. A key procedure of the rite is the construction of a procession, a group of people following a designated leader during the migration. The leader is called the head of the procession; all other people in the processions are called the *followers*, and the last *follower* is the tail (i.e., they have the tail flag set).

Let us show the details of the procession creation and maintenance.

A procession is *created* by a head; initially the head is also the tail of the procession. If the head wants to recruit *settlers* (i.e., people who are still on L_0) then they set the additional follow-me flag. When a *settler* sees a head p with the follow-me flag, they change their state to *prefollower*; furthermore, they memorize the direction, the tail flag, and the previous location of p. At the next round this *prefollower* will try to move in the location where p was seen. If such a move succeeds, they change state to *follower* and update their direction, tail flag, and previous location to the memorized values. If instead the location is occupied by another person p', who must also be part of a procession, the *prefollower* memorizes the direction and tail flag of p' and replicates the aforementioned steps; eventually, the *settler* will be able to leave L_0 and join the procession.

During the whole ritual, it might happen that new *followers* join the procession, hence the tail changes: if a *follower* has the tail flag set, and sees in their previous location a *follower* or a *prefollower* whose direction is pointing at then, then they lose the tail flag and skip the current round.

The procession *moves* according to the following general rule: a *follower* whose direction is not occupied by anyone and whose previous location is occupied moves towards the location pointed by the direction.

There are three exceptions to this rule: (E1) a tail moves regardless of the presence of someone in its previous location; (E2) if a *follower* moved in the previous round and they find that there is no person in their direction, than they change the direction to point to the only *follower* (or head) who is present in a location perpendicular to the old direction, towards L_0. This rule is used by *followers* to move *around* the extremes of L_0, thus allowing the procession to *loop* around the starting line; (E3) the last exception is given by a head that has both follow-me and tail flag set: in this case, when the head sees that there is a *settler* who may join the procession, they move to the next location and they wait for one round (giving time to the *settler* to join the procession).

A special procedure is executed when the head meets a *marker* or someone on their path: in this case, the head becomes a *follower* and a new head is elected (as an example, see Rules 4.F, 4.G of The Chosen One case).

Straight-Line Query. A stationary head can check if their own procession is aligned or not, by sending a *"query message"* to the closest follower in the procession (technically, they send the message to themselves, and then they process this message as a *follower*). Upon reception of the *"query message"*, a *follower* checks if there is a *settler* (or a *prefollower*) in a "non-previous" location who is pointing at them; if so, they wait two rounds. After waiting, they send the *"query message"* to the *follower* in the previous location. The *"query message"* is then propagated in this direction along the procession until it reaches the tail. When the message reaches a tail, they wait using the same rules of the *followers*, and then they send a *"straight message"* back towards the head (again, technically, they start the propagation of this message by sending it to themselves). This message is thus propagated through the *followers* in the procession, with the additional rule that it is changed from *"straight message"* to *"not straight message"* by a *follower* who has both direction and previous location set to vertical. When the head receives the *"straight/not straight message"*, the query terminates.

4 Analysis

We assume that at least $n - f \geq 5$ *settlers* participate in the rite. A settler steps out of the initial line if and only if they become an *explorer* or if they see a procession. Thus we can observe that, if $f = 0$, the *settlers* do not move outside the initial row.

▶ **Lemma 1.** *The* Exodus *sub-rite terminates and it elects an* exodus.leader *if and only if* $f = 0$*. If* n *is even, then two* exodus.leaders *are elected; if* n *is odd, one* exodus.leader *is elected. When the* Exodus *sub-rite terminates the rite specifications are satisfied.*

▶ **Lemma 2.** *If* $f > 0$*, then at the end of the third round we have that either there is at least one* marker *with flag scepter and two* explorers*, or there are at least three* explorers*.*

From the rules of the Explorer Divination sub-rite we have the following corollary.

▶ **Corollary 3.** *At the end of the third round, between the endpoints of the initial line, there cannot be three consecutive empty locations.*

In the following we assume $f > 0$.

▶ **Lemma 4.** *There exists a round* $r \in [3, 4n + 3]$ *in which the rite is in one of the following configurations:*

(C1) *There is a* chosen (The Chosen One).

(C2) *There are two* markers *without sceptre and two* chiefs *on opposite sides of the initial line* (Opposite Sides).

(C3) *There are two* markers *without sceptre and two* chiefs *on the same side of the initial line* (Same Side).

Let a procession be called *pious* if it does not contain two consecutive empty spots and no follower without tail has empty spots both ahead and behind in the procession.

▶ **Lemma 5.** *All the processions generated by* WonnaGo *are pious.*

▶ **Corollary 6.** *Let us consider a* follower *who is not a tail. If at round* r *there is no one in the* follower*'s previous location, then there will be a* follower *at round* $r + 1$*.*

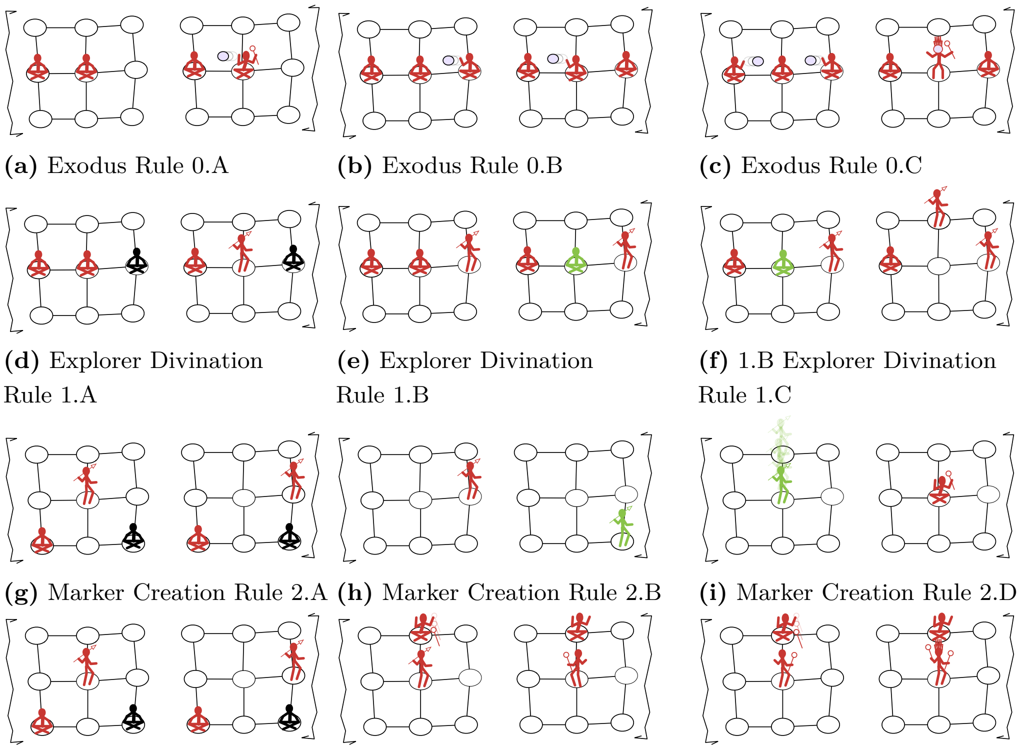
▶ **Corollary 7.** *If there is a unique procession, the head occupies a new location within a constant number of rounds.*

▶ **Theorem 8.** *If a* chosen *is created, then it is unique, and line recovery is achieved within* $\mathcal{O}(n)$ *rounds.*

▶ **Theorem 9.** *Let the tribe reach, at round* r*, a configuration in which there are two* markers *without scepter and two* chiefs *on opposite sides of the initial line. Then, line recovery is achieved within* $\mathcal{O}(n)$ *rounds.*

▶ **Theorem 10.** *Let the tribe reach, at round* r*, a configuration in which there are two* markers *without scepter and two* chiefs *on the same sides of the initial line. Then, line recovery is reached in* $\mathcal{O}(n)$ *rounds.*

Due to Lemma 4 and Theorems 8, 9, 10, line recovery is achieved in $\mathcal{O}(n)$ rounds.



(a) Exodus Rule 0.A (b) Exodus Rule 0.B (c) Exodus Rule 0.C

(d) Explorer Divination Rule 1.A (e) Explorer Divination Rule 1.B (f) 1.B Explorer Divination Rule 1.C

(g) Marker Creation Rule 2.A (h) Marker Creation Rule 2.B (i) Marker Creation Rule 2.D

(j) Marker Creation Rule 2.E (k) Chief Identification Rule 3.A (l) The Chosen one Rule 4.A

Figure 2 Reproduction of paintings depicting specific rules of the WonnaGo.

5 Forming a compact straight line

The WonnaGo ritual solves a convergent task. However, it is also possible to modify the rite to obtain a solution to the stronger *compact line recovery* problem, where an explicit termination is required and the final configuration has to be a compact line (or two compact lines of equal length). To terminate explicitly, the head of a procession has to know when all *settlers* (or half if there exists another procession with opposite direction) have joined the procession and the procession is on a straight line. Knowing this, the head stops moving and, within n rounds, all *followers* will be in a compact position.

6 The Rules

- **Exodus 0:**
  - **Rule 0.A: If** I am a *settler* and I have only one neighbour and it is the first round, **then** I become a *marker* with flag sceptre. Moreover, if my neighbour is non-*sleeper* I ask they "Exodus?".
  - **Rule 0.B:If** I am a *settler* and I receive for first time an "Exodus?" from only one neighbour and my other neighbour is a *settler*, **then** I ask they "Exodus?".
  - **Rule 0.C: If** I am a *settler* and I receive "Exodus?" from both neighbours, **then** I become an *exodus.leader* and I set *dir* to point one of my neighbours.
  - **Rule 0.D: If** I am a *settler* and I receive an "Exodus?" message in two consecutive rounds, **then** I become an *exodus.leader* and I set as *dir* the direction of the first received message and as *pre* the opposite.

- **Rule 0.E: If** I am an *exodus.leader*, **then** I send a “setdir message” to my neighbour pointed by *dir* and a “setbackdir message” to my neighbour pointed by *pre*.
- **Rule 0.F: If** I am a *settler* or a *marker* and I receive a “setdir message” from neighbour p in location x, **then** I set my *dir* to point at the opposite location of x and I send the message to neighbour in location pointed by *dir*.
- **Rule 0.G: If** I am a *settler* or a *marker* and I receive a “setbackdir message” from neighbour p in location x, **then** I set my *dir* to point at location of x and I send the message to neighbour in location opposite to x.

Explorer Divination (1):

- **Rule 1.A: If** I am a *settler* and I have two neighbours and I am near a *sleeper*, **then** I become an *explorer*. My direction *dir* will be pointing at the *sleeper*. If they are both sleepers the tie is broken arbitrary.
- **Rule 1.B:If** it is the second round and I am a *settler* and I have only one *explorer* as neighbour, **then** I become a *settler*.notified. My direction *dir* will be pointing at my *explorer* neighbour.
- **Rule 1.C: If** it is the third round and (I am an *explorer* or (I am a *settler*.notified and I do not have as neighbour a *settler*.notified)), **then** I become *explorer* and I move perpendicularly to my neighbours (up or down).

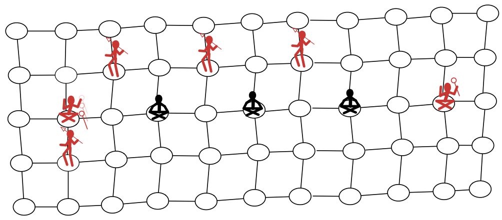
Marker Creation (2):

- **Rule 2.A:If** I am an *explorer* and there is no one in the *dir* direction and there is no (*marker*, *premarker*, *chosen*, *opposer* or *collector*) in my neighbourhood and on the initial line I have not seen three consecutive empty locations, **then** I move to that location.
- **Rule 2.B:If** I am an *explorer* and on the initial line I have seen three consecutive empty locations, **then** I set my state to *premarker* and I move to occupy a free location on L_0.
- **Rule 2.C:If** I am a *premarker* and I did bumped, **then** I become an *explorer*.
- **Rule 2.D: If** I am a *premarker* and I did not bumped, **then** I become a *marker* with flag sceptre.
- **Rule 2.E:If** I am an *explorer* and I see an *explorer* in my direction, **then** I switch the direction *dir* and I skip the current round.

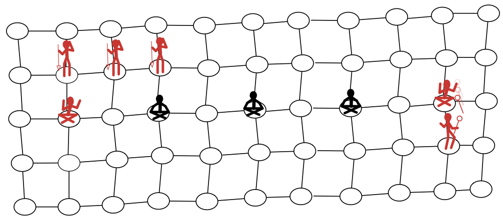
Chief Identification (3):

- **Rule 3.A:If** I am an *explorer* and I see a *marker* with flag sceptre and I receive the sceptre from they, **then** I become a *chief* and I switch the direction *dir*.
- **Rule 3.B:If** I am an *explorer* and I see a *marker* with flag sceptre and I do not receive the sceptre from they, **then** I become a *disciple*.
- **Rule 3.C:If** I am an *explorer* and I see a *marker* without sceptre **then** I become a *disciple*.
- **Rule 3.D:If** I am an *explorer* and there is a *chief* pointed by my *dir* and the *dir* of the *chief* is pointing at me, **then** I become a *chief* and I switch direction *dir*.
- **Rule 3.E:If** I am an *explorer* and there is a *disciple* pointed by *dir*, **then** I become a *disciple*.
- **Rule 3.F:If** I am a *marker* with flag sceptre and I see some *explorers* and I do not see a *chief*, **then** I unset the flag sceptre and I give the sceptre to only one *explorer*.
- **Rule 3.G:If** I am a *chief* and there is an *explorer* pointed by my *dir* and the *dir* of the *explorer* is pointing at me, **then** I become a *disciple*.
- **Rule 3.H:If** I am a *chief* and there is a *disciple* pointed by my *dir*, **then** I become a *disciple*.

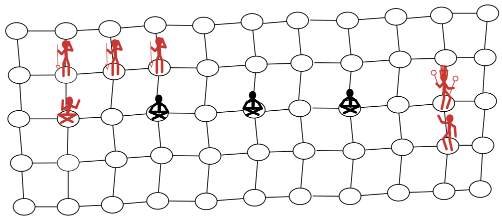
- **Rule 3.I:If** I am a *disciple* and there is a neighbour *chief* whose direction *dir* is pointing a me, **then** I become a *chief* and my direction *dir* will be the one of the old *chief*.



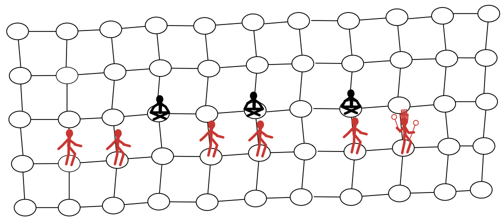
(a) Chosen one: Election of the *chief* (Rules 3.A, 3.F).



(b) Chosen one: Election of the *chosen* (Rules 4.A, 4.I).



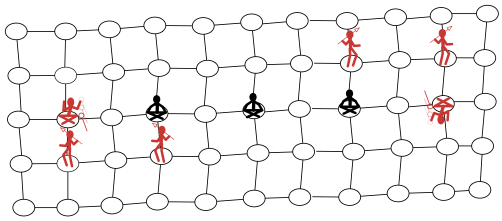
(c) Chosen one: The *chosen* changes side (Rules 4.G, 4.F).



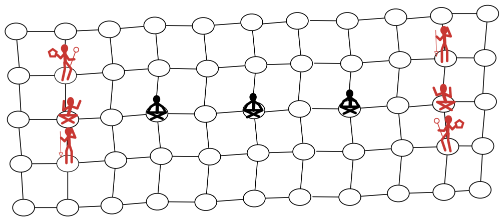
(d) Chosen one: The *chosen* leads the migration.

Figure 3 Reproduction of paintings depicting a rite in which a *chosen* is elected.

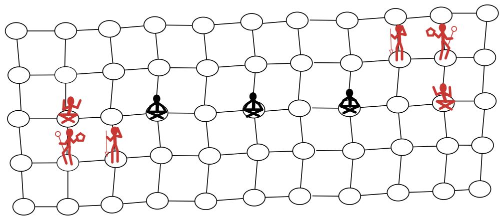
- **The Chosen one (4):**
  - **Rule 4.A:If** I am a *chief* and I see a *marker* with flag sceptre, **then** I become the *chosen*, I set the follow-me flag and the tail flag and I skip the round.
  - **Rule 4.B:If** I am *chosen* and there is a person pointed by *dir*, **then** I become a *follower* and I unset the follow-me flag.
  - **Rule 4.C:If** direction *dir* of a *chosen* is pointing at me, **then** I copy the state of the *chosen* and I set direction *prev* to point the location the old *chosen*.
  - **Rule 4.D:If** I am *chosen* and I do not see a *marker* and in my direction *dir* there is no one and my direction *dir* is not pointing to a location outside L_1, L_0, L_{-1}, **then** I move to that location.
  - **Rule 4.E:If** I am *chosen* and I do not see a *marker* and in my direction *dir* there is no one and my direction *dir* pointing to a location outside L_1, L_0, L_{-1}, **then** I switch *dir* to point towards the farther endpoint of the line.
  - **Rule 4.F:If** I am *chosen* and I am a neighbour of a *marker* without sceptre, **then**: I become *follower* and I set *dir* to point at the *marker*.
  - **Rule 4.G:If** I am a *marker* and I am a neighbour of a *chosen*, **then** I copy the state of the *chosen* and I set direction *prev* to point at location the old *chosen* and as direction *dir* the opposite location.
  - **Rule 4.H:If** I am a *chief* and on the line I see three consecutive empty locations, **then**: I set my state to *chosen*, I set the follow-me flag and the tail flag and I set as direction *dir* the only location in L_0.
  - **Rule 4.I:If** I am a *marker* with flag sceptre and I see a *chief*, **then** I unset the flag sceptre.
- **Opposite Sides (5):**
  - **Rule 5.A: If** I am a *chief* and I reach a *marker* M without sceptre, **then** I become a *collector*, I set the follow-me and tail flags, and I switch direction *dir*.



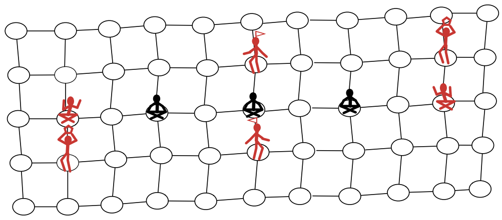
(a) Opposite Sides: Two *chiefs* are elected on the opposite side (Rules 3.A, 3.F).



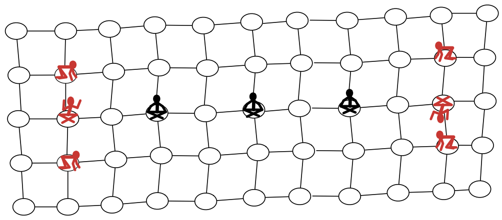
(b) Opposite Sides: The two *chiefs* become *collector*.



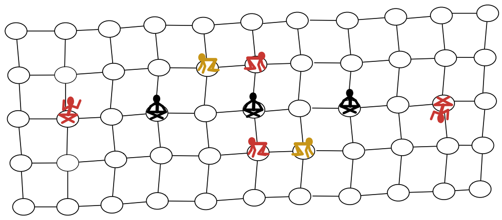
(c) Opposite Sides: Two *collector*s order the *followers* to become mover (Rule 5.E).



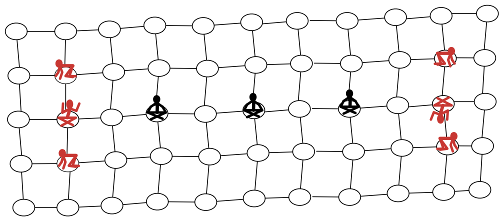
(d) Opposite Sides: The *movers* are going towards the other *marker* (Rule 5.H).



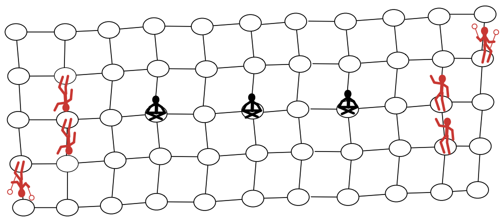
(e) Opposite Sides: The *probes* start moving on both sides of the line (Rule 5.N). Each *marker* memorised the side of the *collector.counting* (Rule 5.K).



(f) Opposite Sides: Bumping *probes*. Two *probes* have not bumped (colored in red), they immediately move (5.P). The *probes* in yellow have bumped, they have $probecounter = 1$ and they will decrease their counters in this round (Rules 5.O, 5.M).



(g) Opposite Sides: The *probes* reach the *marker* at same time on both sides. This triggers (Rule 5.U).



(h) Opposite Sides: Two processions with equal size are migrating in opposed directions.

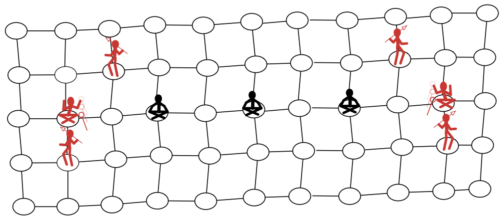
Figure 4 Reproduction of paintings depicting a rite in which the Opposite Sides case arise.

- **Rule 5.B: If** I am a *collector* and there is no one in the direction of *dir* and I do not see a *marker* and the round is even, **then** I move.
- **Rule 5.C: If** I am a *disciple* and the *dir* of a *collector* is pointing at me and the round is even, **then** I become a *collector* and I copy the directions $dir, prev$ and follow-me flag of old *collector* but not the tail flag.
- **Rule 5.D: If** I am a *collector* and my *dir* is pointing at *disciple* and the round is even,

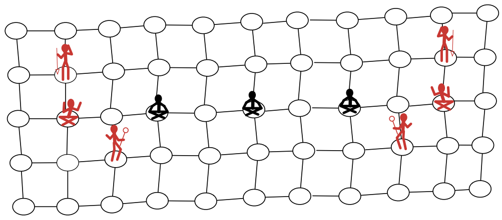
then I become a *follower*, if the tail flag is set I retain it.

- **Rule 5.E: If** I am a *collector* and I have reached the *marker* M', **then** I wait for one round then I signal to the *follower* pointed by *prev* to become a *mover* and after waiting two rounds I become a *collector.counting*. If there is no such *follower*, then I become a *collector .counted* and I switch the direction in *dir*.
- **Rule 5.F: If** I am a *mover* and my *dir* is pointing at a *follower*, **then** I become a *follower* and I copy the state of the pointed *follower*.
- **Rule 5.G: If** I am a *follower* and the *dir* of a *mover* is pointing at me, **then** I become a *mover* and I copy the *dir* of the old *mover*.
- **Rule 5.H: If** I am a *mover* or a *collector .counted* and there is no one pointed by *dir* and I do not see a *marker*, **then** I move in the direction.
- **Rule 5.I: If** I am a *marker* and I see a *collector .counted* and a *collector .counting*, **then** I signal the victory to *collector .counted*.
- **Rule 5.J: If** I am *collector .counted* and I receive the victory signal, **then** I become a *consul* and I point the *marker*. A *consul* obeys to the same rules of the *chosen*. If somebody see a *consul* they do as if they see a *chosen*.
- **Rule 5.K: If** I am a *marker* and I see a *mover* and a *collector .counting*, **then** I send a probing signal to both and I memorise the side of the *collector .counting* in *dir* for future reference.
- **Rule 5.L: If** I am a *mover* or a *collector .counting* and I receive a probing signal from the marker, **then** I become a *probe* and I set $probecounter = 1$ and I switch direction.
- **Rule 5.M: If** I am *probe* with $probecounter \geq 1$, and there is no *probe* whose *dir* is pointing at me with $probecounter = 0$, **then** I decrease my *probecounter*.
- **Rule 5.N: If** I am a *probe* and my *probecounter* is 0, and in there is no one in the direction pointed by *dir*, **then** I move in my direction.
- **Rule 5.O: If** I am a *probe* and I see a *probe* whose *dir* is pointing at me, and one of us has $probecounter = 0$, **then** I take the state of the other one, I add 1 to *probecounter* and I further add one to *probecounter* if I see a *prefollower* whose *dir* is pointing at me. Moreover, If I bumped then I set a visible bumped flag in my state.
- **Rule 5.P: If** I am a *probe* and behind me there is a *probe* with visible bumped flag set, **then** I decrease my *probecounter*, If the *probecounter* is 0 then I move in the direction of *dir*.
- **Rule 5.Q: If** I am a *probe*, my $probecounter = 0$ in my direction *dir* there is a person that is not a *probe*, **then** I copy the state of the other.
- **Rule 5.R: If** I am not a *probe*, and there is only one *probe* whose *dir* is pointing at me with $probecounter = 0$, **then** I become a *probe* I copy the direction *dir* from the old probe and I set $probecounter = 1$ and I further add one to *probecounter* if I see a *prefollower* pointing at me.
- **Rule 5.S: If** I am not a *probe*, and there are two *probes* whose *dir* are pointing at me with $probecounter = 0$, **then** I wait one round and a further round if I see a *prefollower* whose *dir* is pointing at me. Then, I order each of my neighbours to become a *probe* with *dir* pointing away from me and set their *probecounter* to 2 or 1, respectively if they see a *prefollower* pointing at them or not. At the next round I take my original non-*probe* state.
- **Rule 5.T: If** I am *probe* and I have reached a *marker* and my $probecounter = 0$, **then** I become a *follower* and I set *dir* to point the *marker*.
- **Rule 5.U: If** I am a *marker*, the first *probe* that reaches me and has $probecounter = 0$ is on the side I memorised in Rule 5.K **then**

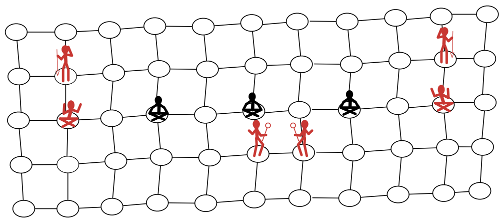
  * if on the other side there is a *probe* p that has *probecounter* > 0 or there is no one, **then** I become a *consul* and I point in the direction of p. A *consul* obeys to the same rules of the *chosen*. If somebody see a *consul* they do as if they see a *chosen*.
  * if on the other side there is a *probe* p that has *probecounter* $= 0$, **then** I order p to become *consul* pointing in the direction opposite to the other *marker*, and I become *follower* pointing at p and setting as *pre* the opposite.



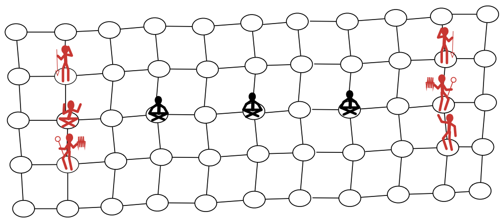
(a) Same Side: Two *chiefs* are elected on the same side (Rules 3.A, 3.F).



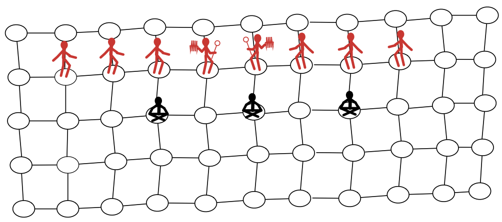
(b) Same Side: Two *chiefs* are moving towards each other.



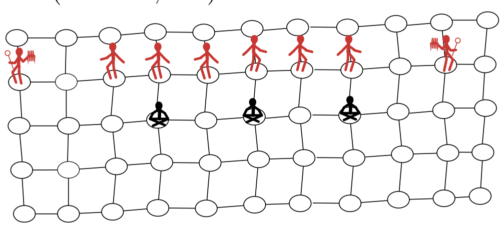
(c) Same Side: Two *chiefs* met (Rule 6.A).



(d) Same Side: Two *opposers* are changing side (Rule 6.F, 6.G).



(e) Same Side: Two *opposer* meet. They are executing the Straight Line Query.



(f) Same Side: Migration of two processions of equal size.

Figure 5 Reproduction of paintings depicting a rite in which the Same Side case arise.

- **Same Side (6):**
  - **Rule 6.A: If** I am a *chief* and there is another *chief* pointed by me, **then**: I become *opposer*, I set the tail flag and I switch direction.
  - **Rule 6.B:If** I am an *explorer* and there is a *opposer* pointed by me and the *opposer* is pointing at me, **then** I become an *opposer* and I switch direction for *dir* and I set *prev* to point at the location the old *opposer*.
  - **Rule 6.C:If** I am a *opposer* and there is an *explorer* pointed by me and the *explorer* is pointing at me, **then** I become a *follower*.
  - **Rule 6.D:If** I am *opposer* and I see a *disciple*, **then** I become a *follower*.
  - **Rule 6.E:If** I am a *disciple* and I see only one *opposer*, **then** I become the *opposer* and I set *prev* to point at the old *opposer* and *dir* to point at the location of the neighbour *disciple* if they exist otherwise the free location closer to the initial line.

- **Rule 6.F:If** I am a *marker* and I see an *opposer*, **then** I become an *opposer*, I set the follow-me flag and I set *prev* to point at the old *opposer* and *dir* as the opposite.
- **Rule 6.G:If** I am *opposer* and I see a *marker*, **then** I become a *follower* and I point at the *marker*.
- **Rule 6.H:If** I am *opposer* and I see another *opposer* pointed by me for two consecutive rounds, **then** I do a query in my procession and I set my state to *opposer.counting*.
- **Rule 6.I:If** I am an *opposer.counting* and I see a *opposer.winner*, **then** I become a *follower*.
- **Rule 6.J:If** I am an *opposer.counting* and my query terminates and the result is "straight" and I do not see an *opposer.winner*, **then** I become an *opposer.winner*.
- **Rule 6.K:If** I am an *opposer.counting* and my query terminates and the result is "not-straight" and I do not see an *opposer.winner*, **then** I become an *opposer.winner*.
- **Rule 6.L:If** I am an *opposer.winner* and the previous round I was an *opposer.counting* and I have as neighbour an *opposer.winner*, **then** if we are both "straight" I set the tail flag and I switch direction *dir*, if I'm the only one "straight" I switch direction *dir*, if I'm not "straight" I become a *follower*.
- **Rule 6.M:If** I am a *follower* and there is an *opposer.winner* that is pointing at me, **then** I become a *opposer.winner* and I switch direction *dir*, I clear my tail flag, and I set *prev* to point to the old *opposer.winner*.
- **Rule 6.N:If** I am *opposer.winner* and there is a *follower* pointed by me, **then** I become a *follower* and I point the *follower*.
- **Rule 6.O:If** I am a *disciple* and I see two *opposers*, **then** I become an *opposer.winner*.
- **Rule 6.P:If** I am a *opposer* there is no one in the *dir* direction and I do not see a *marker*, **then** I move.

- **Procession (7):**
  - **Rule 7.A: If** I am a *follower* without tail flag and I see a *follower* in my previous position and I do not see anyone in *dir* direction, **then** I move towards my direction.
  - **Rule 7.B: If** I am a *settler* or a *settler*.notified and I see an head with follow-me flag in location l, **then** I set my state to *prefollower*; from the person in l I copy their tail flag and their *dir* direction in my *pre*, and I point towards l.
  - **Rule 7.C: If** I am a *prefollower* and I bumped and there is no one in direction *dir*, **then** I move towards *dir*.
  - **Rule 7.D: If** I am a *prefollower* and I bumped and there is someone in direction *dir* , **then** I copy the tail flag from the person in *dir*.
  - **Rule 7.E: If** I am a *prefollower* and I moved, **then** I set my state to *follower* and I set my *dir* to be the *pre* and *pre* as the opposite of *dir*, the queue flag is set to the memorised value.
  - **Rule 7.F: If** I have a tail flag set and in the location pointed by *pre* there is a *prefollower* or *follower* neighbour pointing at me, **then** I unset my tail flag.
  - **Rule 7.G: If** I am a *follower* and I moved in the previous round and I do not see a *follower* or an head in direction *dir*, **then** I set *dir* to point towards the only head or *follower* perpendicular to my old *dir*.
  - **Rule 7.H:If** I am the head of a *followers* procession and I have the tail flag set and one of my neighbours is a *settler*, **then** I set the waiting flag and I move in my direction.
  - **Rule 7.I:If** I am the head of a *followers* procession, and at the previous round I had set the waiting flag, **then** I reset the waiting flag and I do not move in this round.
  - **Rule 7.J: If** I am the head of a *followers* procession, I do not have the tail flag, and I do not see a *follower* in my previous position, **then** I do not move.

- **Rule 7.K:** **If** I am a *prefollower* and I bumped and in direction *dir* there is a *probe*, **then**, I skip this round.

- **Straight Line Query (8):**
  - **Rule 8.A:** If I receive a "query message" and there is no *settler* (or a *prefollower*) pointing at me, **then** if I am a tail, I send the "straight message" to myself. Otherwise I forward the "query message" at the person pointed by *pre*.
  - **Rule 8.B:** If I receive a "query message" and there is a *settler* (or a *prefollower*) pointing at me, **then** I skip two rounds. Then if I am a tail I send the "straight message" to myself. Otherwise I forward the "query message" at the person pointed by *pre*.
  - **Rule 8.C:** If I receive a "straight message", **then** If I my *dir* and *pre* are vertical I forward a "not-straight message". Otherwise, I forward "straight message".
  - **Rule 8.D:** If I receive a "not-straight message", **then** I forward a "not-straight message".
  - **Rule 8.E:** If I am an *opposer* and I receive a "not-straight message", **then** my query is terminated and the result is "not-straight".
  - **Rule 8.F:** If I am an *opposer* and I receive a "straight message", **then** my query is terminated and the result is "straight".

7 Conclusion

The ritual that we have analyzed in this paper is the *short* WonnaGo. The cave contains other paintings depicting a more complex version of the rite known as the *long* WonnaGo. The long rite was used in the autumnal months, when the new flowers of Cannabis have the greatest quantity of THC. It is very likely that the Astracinca had knowledge of the temporal distortion caused by phytocannabinoids, an effect studied only recently by modern medicine [2]. Essentially, the long ritual takes into account that different settlers may have a different perception of the passage of time, and that furthermore they can unpredictably slow down both in their movements and communication. We are studying the *long* WonnaGo, and we are confident that the Astracinca devised an algorithm for *line recovery* in the *asynchronous* settings. However, we have yet to fully understand the many subtleties of the *long* WonnaGo, which remains a challenging open problem.

References

1. N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006.
2. Z. Atakan, P. Morrison, M.G. Bossong, R. Martin-Santos, and J.A. Crippa. The effect of cannabis on perception of time: a critical review. *Current Pharmaceutical Design*, 18(32):4915–22, 2012.
3. D.G. Barceloux. *Medical Toxicology of Drug Abuse*. Wiley, 2012.
4. L. Barriere, P. Flocchini, E. Mesa-Barrameda, and N. Santoro. Uniform scattering of autonomous mobile robots in a grid. *International Journal of Foundations of Computer Science*, 22(3):679–697, 2011.
5. Z.J. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *International Journal of Robotics Research*, 23(9):919–937, 2004.
6. G. Chirikjian. Kinematics of a metamorphic robotic system. In *Proceedings of the 1994 International Conference on Robotics and Automation (ICRA)*, pages 449–455, 1994.

7 M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.

8 Z. Derakhshandeh, S. Dolev, R. Gmyr, A. Richa, C. Scheideler, and T. Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of the 2nd International Conference on Nanoscale Computing and Communication*, pages 21:1–21:2, 2015.

9 Z. Derakhshandeh, R. Gmyr, T. Strothmann, R.A. Bazzi, A. Richa, and C. Scheideler. Leader election and shape formation with self-organizing programmable matter. In *Proceedings of the 21st International Conference on DNA Computing and Molecular Programming (DNA)*, pages 117–132, 2015.

10 Y. Emek, T. Langner, J. Uitto, and R. Wattenhofer. Solving the ANTS problem with asynchronous finite state machines. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, 471–482, 2014.

11 P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Ring exploration by asynchronous oblivious robots. *Algorithmica*, 65(3):562–583, 2013.

12 P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Morgan & Claypool, 2012.

13 T.R. Hsiang, E. Arkin, M. Bender, S. Fekete, and J. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Proceedings of the 5th Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 77–94, 2002.

14 J.E. Joy, S.J. Watson, Jr., and J.A. Benson. *Marijuana and medicine: assessing the science base*. National Academies Press, 1999.

15 B. Keller, T. Langner, J. Uitto and R. Wattenhofer. Overcoming obstacles with ants. In *Proceedings of the 19th International Conference on Priniciples of Distributed Systems (OPODIS)*, 2015.

16 R. Nozick. *Anarchy, State, and Utopia*. Basic Books, 1974.

17 A.L. Rosenberg. Finite-state robots in the land of Rationalia. In *Proceedings of the 27th International Symposium on Computer and Information Sciences*, 3–11, 2013.

18 G. Samorini. *Animals and Psychedelics: The Natural World and the Instinct to Alter Consciousness*. Park Street Press, 2002.

19 I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

20 J.E. Walter, J.L. Welch, and N.M. Amato. Distributed reconfiguration of meta-morphic robot chains. *Distributed Computing*, 17(2):171–189, 2004.