On Mobile Agent Verifiable Problems

Evangelos Bampas, David Ilcinkas

On Mobile Agent Verifiable Problems*

Evangelos Bampas and David Ilcinkas

CNRS & Univ. Bordeaux, LaBRI, UMR 5800, F-33400, Talence, France
{evangelos.bampas, david.ilcinkas}@labri.fr

Abstract. We consider decision problems that are solved in a distributed fashion by synchronous mobile agents operating in an unknown, anonymous network. Each agent has a unique identifier and an input string and they have to decide collectively a property which may involve their input strings, the graph on which they are operating, and their particular starting positions. Building on recent work by Fraigniaud and Pelc [LATIN 2012, LNCS 7256, pp. 362–374], we introduce several natural new computability classes allowing for a finer classification of problems below co-MAV or MAV, the latter being the class of problems that are verifiable when the agents are provided with an appropriate certificate. We provide inclusion and separation results among all these classes. We also determine their closure properties with respect to set-theoretic operations. Our main technical tool, which is of independent interest, is a new meta-protocol that enables the execution of a possibly infinite number of mobile agent protocols essentially in parallel, similarly to the well-known dovetailing technique from classical computability theory.

1 Introduction

1.1 Context and motivation

The last few decades have seen a surge of research interest in the direction of studying computability- and complexity-theoretic aspects for various models of distributed computing. Significant examples of this trend include the investigation of unreliable failure detectors [5,6], as well as wait-free hierarchies [14]. A more recent line of work studies the impact of randomization and non-determinism in what concerns the computational capabilities of the $\mathcal{LOCAL}$ model [9,12], as well as the impact of identifiers in the same model [10,11]. A different approach considers the characterization of problems that can be solved under various notions of termination detection or various types of knowledge about the network in message-passing systems [1,2,3,4,17]. Finally, a recent work focuses on the computational power of teams of mobile agents [13]. Our work lies in this latter direction.

* This work was partially funded by the ANR projects DISPLEXITY (ANR-11-BS02-014) and MACARON (ANR-13-JS02-002). This study has been carried out in the frame of the "Investments for the future" Programme IdEx Bordeaux – CPU (ANR-10-IDEX-03-02).

The mobile agent paradigm has been proposed since the 90's as a concept that facilitates several fundamental networking tasks including, among others, fault tolerance, network management, and data acquisition [15], and has been of significant interest to the distributed computing community (see, e.g., the recent surveys [7,16]). As such, it is highly pertinent to develop a computability theory for mobile agents, that classifies different problems according to their degree of (non-)computability, insofar as we are interested in really understanding the computational capabilities of groups of mobile agents.

In this paper, we consider a distributed system in which computation is performed by one or more deterministic mobile agents, operating in an unknown, anonymous network. Each agent has a unique identifier and is provided with an input string, and they have to collectively decide a property which may involve their input strings, the graph on which they are operating, and their particular starting positions. One may argue about the usefulness of developing a theory specifically for mobile agent decision problems. We believe that, apart from its inherent theoretical interest, such a study is bound to yield intermediate results, tools, intuitions, and techniques that will prove useful when one moves on to consider from a computability/complexity point of view other, perhaps more traditional, mobile agent problems, such as exploration, rendezvous, pattern formation, etc. One such tool is the protocol that we develop in this paper, which enables the interleaving of the executions of a possibly infinite number of mobile agent protocols.

1.2 Related work

In [13], Fraigniaud and Pelc introduced two natural computability classes, MAD and MAV, as well as their counterparts co-MAD and co-MAV. The class MAD, for "Mobile Agent Decidable", is the class of all mobile agent decision problems which can be *decided*, i.e., for which there exists a mobile agent protocol such that all agents accept in a "yes" instance, while at least one agent rejects in a "no" instance. On the other hand, the class MAV, for "Mobile Agent Verifiable", is the class of all mobile agent decision problems which can be *verified*. More precisely, in a "yes" instance, there exists a certificate such that if each agent receives its dedicated piece of it, then all agents accept, whereas in a "no" instance, for every possible certificate, at least one agent rejects. Certificates are for example useful in applications in which repeated verifications of some property are required. Fraigniaud and Pelc proved in [13] that MAD is strictly included in MAV, and they exhibited a problem which is complete for MAV under an appropriate notion of oracle reduction.

In [8], Das et al. focus on the complexity of distributed verification, rather than on its computability. In fact, their model differs in several aspects. First of all, the networks in which the mobile agents operate are not anonymous, but each node has a unique identifier. This greatly facilitates symmetry breaking, a central issue in anonymous networks. On the other hand though, the memory of the mobile agents is limited. Indeed, in [8], the authors study the minimal amount of memory needed by the mobile agents to distributedly verify some

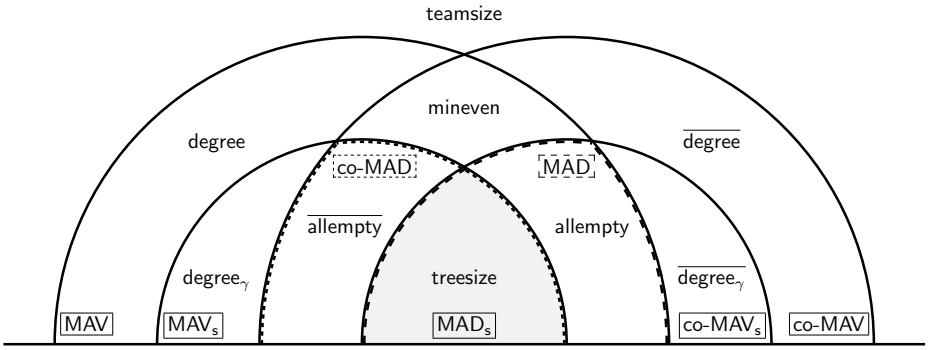

Fig. 1. Containments between classes below MAV and co-MAV with corresponding illustrative problems. Class and problem definitions are summarized in Tables 1 and 2, respectively.

classes of graph properties. Again, the studied properties are different from the ones studied here and in [13], since they do not depend on the mobile agents or their starting positions. However, they may depend on labels that nodes can possess in addition to their unique identifiers.

1.3 Our contributions

We introduce several new mobile agent computability classes which play a key role in our endeavor for a finer classification of problems below MAV and co-MAV. The classes MAD_s and MAV_s are strict versions of MAD and MAV, respectively, in which unanimity is required in both "yes" and "no" instances. Furthermore, we consider the class co-MAV′ (and its counterpart MAV′) of mobile agent decision problems that admit a certificate for "no" instances, while retaining the system-wide acceptance mechanism of MAV.

We perform a thorough investigation of the relationships between the newly introduced and pre-existing classes. As a result, we obtain a complete Venn diagram (Figure 1) which illustrates the tight interconnections between them. We take care to place natural decision problems (in the mobile agent context) in each of the considered classes. Among other results, we obtain a couple of fundamental, previously unknown, inclusions which concern pre-existing classes: MAD $\subseteq$ co-MAV and co-MAD $\subseteq$ MAV.

We complement our results with a complete study of the closure properties of these classes under the standard set-theoretic operations of union, intersection, and complement. The various class definitions together with the corresponding closure properties are summarized in Table 1.

The main technical tool that we develop and use in the paper is a new meta-protocol that enables the execution of a possibly infinite number of mobile agent protocols essentially in parallel. This can be seen as a mobile agent computing analogue of the well-known dovetailing technique from classical recursion theory.

Proofs are omitted due to lack of space.

Table 1. Overview of mobile agent decidability and verifiability classes and their closure properties. The notation yes (resp. no) means that all agents accept (resp. reject). Similarly, $\widehat{\mathsf{yes}}$ (resp. $\widehat{\mathsf{no}}$) means that at least one agent accepts (resp. rejects).

	Definition				*Closure Properties*		
	"yes" instances		"no" instances		Union	Intersec.	Compl.
$\mathsf{MAD_s}$	($\forall$ certificate:)	yes	($\forall$ certificate:)	no	✓	✓	✓
MAD	($\forall$ certificate:)	yes	($\forall$ certificate:)	$\widehat{\mathsf{no}}$	✗	✓	✗
co-MAD	($\forall$ certificate:)	$\widehat{\mathsf{yes}}$	($\forall$ certificate:)	no	✓	✗	✗
$\mathsf{MAV_s}$	$\exists$ certificate:	yes	$\forall$ certificate:	no	✓	✓	✗
$\mathsf{co\text{-}MAV_s}$	$\forall$ certificate:	yes	$\exists$ certificate:	no	✓	✓	✗
MAV	$\exists$ certificate:	yes	$\forall$ certificate:	$\widehat{\mathsf{no}}$	✗	✓	✗
co-MAV	$\forall$ certificate:	$\widehat{\mathsf{yes}}$	$\exists$ certificate:	no	✓	✗	✗
MAV′	$\exists$ certificate:	$\widehat{\mathsf{yes}}$	$\forall$ certificate:	no	✓	✓	✗
co-MAV′	$\forall$ certificate:	yes	$\exists$ certificate:	$\widehat{\mathsf{no}}$	✓	✓	✗

2 Preliminaries

The graphs in which the mobile agents operate are undirected, connected, and anonymous. The edges incident to each node v (ports) are assigned distinct local port numbers (also called labels) from $\{1, \ldots, d_v\}$, where d_v is the degree of node v. The port numbers assigned to the same edge at its two endpoints do not have to be in agreement.

We conventionally fix a binary alphabet $\Sigma = \{0, 1\}$. In view of the natural bijection between binary strings and $\mathbb{N}$ which maps a string to its rank in the quasi-lexicographic order of strings (shorter strings precede longer strings, the rank of the empty string ε being 0), we will occasionally treat strings and natural numbers interchangeably. If x and y are strings, then $\langle x, y\rangle$ stands for any standard encoding as a string of the pair of strings (x, y).

If $\boldsymbol{x}$ is a list, then $|\boldsymbol{x}|$ is the length of $\boldsymbol{x}$ and x_i is the i-th element of $\boldsymbol{x}$. If f is a function that can be applied to the elements of $\boldsymbol{x}$, then we will use the notation $f(\boldsymbol{x}) = \big(f(x_1), \ldots, f(x_{|\boldsymbol{x}|})\big)$. In the same spirit, if $\boldsymbol{x}$ and $\boldsymbol{y}$ are equal-length lists of strings, then $\langle \boldsymbol{x}, \boldsymbol{y}\rangle$ stands for the list $\big(\langle x_1, y_1\rangle, \ldots, \langle x_{|\boldsymbol{x}|}, y_{|\boldsymbol{y}|}\rangle\big)$.

We denote by Σ_1^0 the set of recursively enumerable (or Turing-acceptable) decision problems, $\Pi_1^0 = \mathsf{co}\text{-}\Sigma_1^0$, and $\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$. Δ_1^0 is exactly the set of Turing-decidable problems.

2.1 Mobile agent computations

A *mobile agent protocol* is modeled as a deterministic Turing machine. *Mobile agents* are modeled as instances of a mobile agent protocol (i.e., copies of the corresponding deterministic Turing machine) which move in an undirected, connected, anonymous graph with port labels. Each mobile agent is provided initially with two input strings: its ID, denoted by id, and its input, denoted by x.

By assumption, in any particular execution of the protocol, the ID of each agent is unique. The execution of a group of mobile agents on a graph G proceeds in synchronous steps. At the beginning of each step, each agent is provided with an additional input string, which contains the following information: (i) the degree of the current node u, (ii) the port label at u through which the agent arrived at u (or ε if the agent is in its first step or did not move in the previous step), and (iii) the configuration of all other agents which are currently on u. Then, each agent performs a local computation and eventually halts by accepting or rejecting, or it moves through one of the ports of u, or remains at the same node. We assume that all local computations take the same time and that edge traversals are instantaneous. Therefore, the execution is completely synchronous.

Let M be a mobile agent protocol, G be a graph, $\mathbf{id}$ be a list of distinct IDs, $\boldsymbol{s}$ be a list of nodes of G, and $\boldsymbol{x}$ be a list of strings such that $|\mathbf{id}| = |\boldsymbol{s}| = |\boldsymbol{x}| = k > 0$. We denote by $M(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$ the execution of k copies of M, the i-th copy starting on node s_i and receiving as inputs the ID id_i and the string x_i. The tuple $(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$ is called the *implicit input*. Similarly, we denote by $M(\mathsf{id}, x; \mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$ the personal view of the execution of M on the implicit input, as experienced by the agent with ID id and input x. We distinguish between the *explicit* input (id, x), which is provided to the agent at the beginning of the execution, and the implicit input, which may or may not be discovered by the agent in the course of the execution.

Given an implicit input, we write $M(\mathsf{id}, x; \mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x}) = \mathsf{yes}$ (resp. no) if the agent with explicit input (id, x) accepts (resp. rejects) during $M(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$. Furthermore, we write $M(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x}) \mapsto \mathsf{yes}$ (resp. no), if $\forall i\ M(\mathsf{id}_i, x_i; \mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x}) = \mathsf{yes}$ (resp. no), and $M(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x}) \mapsto \widehat{\mathsf{yes}}$ (resp. $\widehat{\mathsf{no}}$), if all agents halt and for some i $M(\mathsf{id}_i, x_i; \mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x}) = \mathsf{yes}$ (resp. no).

2.2 Mobile agent decision problems

Definition 1 ([13]). *A* mobile agent decision problem *on anonymous graphs is a set Π of instances $(G, \boldsymbol{s}, \boldsymbol{x})$, where G is a graph, $\boldsymbol{s}$ is a non-empty list of nodes of G, and $\boldsymbol{x}$ is a list of strings with $|\boldsymbol{x}| = |\boldsymbol{s}|$, which satisfies the following closure property: For every G and for every automorphism α of G that preserves port numbers, $(G, \boldsymbol{s}, \boldsymbol{x}) \in \Pi$ if and only if $(G, \alpha(\boldsymbol{s}), \boldsymbol{x}) \in \Pi$.*[1]

We will refer to instances which belong to a problem Π as "yes" instances of Π. Instances that do not belong to Π will be called "no" instances of Π. The *complement* $\overline{\Pi}$ of a mobile agent decision problem Π is the problem $\overline{\Pi} = \{(G, \boldsymbol{s}, \boldsymbol{x}) : |\boldsymbol{s}| = |\boldsymbol{x}| \text{ and } (G, \boldsymbol{s}, \boldsymbol{x}) \notin \Pi\}$.[2] Some examples of decision problems are shown in Table 2.

[1] Note that this closure property is syntactically different from the one used in [13] due to notational differences, but the two are equivalent.

[2] It is easy to check that if Π is a decision problem, then $\overline{\Pi}$ also satisfies the closure property of Definition 1. Therefore, $\overline{\Pi}$ is also a decision problem.

Table 2. Definitions of some mobile agent decision problems that we use in the rest of the paper.

alone	$= \{(G, \boldsymbol{s}, \boldsymbol{x}) : \lvert\boldsymbol{s}\rvert = 1\}$
allempty	$= \{(G, \boldsymbol{s}, \boldsymbol{x}) : \forall i\ x_i = \varepsilon\}$
consensus	$= \{(G, \boldsymbol{s}, \boldsymbol{x}) : \forall i, j\ x_i = x_j\}$
degree	$= \{(G, \boldsymbol{s}, \boldsymbol{x}) : \forall i\ \exists v\ d_v = x_i\}$
degree_γ	$= \{(G, \boldsymbol{s}, \boldsymbol{x}) : G$ contains a node of degree $\gamma\}$ (for $\gamma \geq 1$)
mineven	$= \{(G, \boldsymbol{s}, \boldsymbol{x}) : \min_i x_i$ is even$\}$
path	$= \{(G, \boldsymbol{s}, \boldsymbol{x}) : G$ is a path$\}$
teamsize	$= \{(G, \boldsymbol{s}, \boldsymbol{x}) : \forall i\ x_i = \lvert\boldsymbol{s}\rvert\}$
treesize	$= \{(G, \boldsymbol{s}, \boldsymbol{x}) : \forall i\ G$ is a tree of size $x_i\}$

Definition 2 ([13]). *A decision problem Π is* mobile agent decidable *if there exists a protocol M such that for all instances $(G, \boldsymbol{s}, \boldsymbol{x})$: if $(G, \boldsymbol{s}, \boldsymbol{x}) \in \Pi$ then $\forall \mathbf{id}\ M(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x}) \mapsto \mathsf{yes}$, whereas if $(G, \boldsymbol{s}, \boldsymbol{x}) \notin \Pi$ then $\forall \mathbf{id}\ M(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x}) \mapsto \widehat{\mathsf{no}}$. The class of all decidable problems is denoted by* MAD.

Definition 3 ([13]). *A decision problem Π is* mobile agent verifiable *if there exists a protocol M such that for all instances $(G, \boldsymbol{s}, \boldsymbol{x})$: If $(G, \boldsymbol{s}, \boldsymbol{x}) \in \Pi$ then $\exists \boldsymbol{y}\ \forall \mathbf{id}\ M(\mathbf{id}, G, \boldsymbol{s}, \langle \boldsymbol{x}, \boldsymbol{y} \rangle) \mapsto \mathsf{yes}$, whereas if $(G, \boldsymbol{s}, \boldsymbol{x}) \notin \Pi$ then $\forall \boldsymbol{y}\ \forall \mathbf{id}\ M(\mathbf{id}, G, \boldsymbol{s}, \langle \boldsymbol{x}, \boldsymbol{y} \rangle) \mapsto \widehat{\mathsf{no}}$. The class of all verifiable problems is denoted by* MAV.

When there is no room for confusion, we will use the term *certificate* both for the string y provided to an agent and for the collection of certificates $\boldsymbol{y}$ provided to the group of agents. If we need to distinguish between the two, we will refer to $\boldsymbol{y}$ as a *certificate vector*. Finally, if X is a class of mobile agent decision problems, then $\mathsf{co\text{-}X} = \{\Pi : \overline{\Pi} \in \mathsf{X}\}$.

Remark 1. Note that in [13], only decidable (in the classical sense) mobile agent decision problems were taken into consideration. As a result, it was by definition the case that MAD and MAV were both subsets of Δ_1^0. For the purposes of this work, we do not impose this constraint.

3 Mobile Agent Decidability Classes

A problem Π is in co-MAD if and only if it can be decided by a mobile agent protocol in a sense which is dual to that of Definition 2: If the instance is in Π, then at least one agent must accept, whereas if the instance is not in Π, then all agents must reject. We will consider one more such variant in the form of the "strict" class $\mathsf{MAD_s}$. A problem belongs to this class if it can be solved in such a way that every agent always outputs the correct answer.

Definition 4. *A decision problem Π is in* $\mathsf{MAD_s}$ *if and only if there exists a protocol M such that for all instances $(G, \boldsymbol{s}, \boldsymbol{x})$: if $(G, \boldsymbol{s}, \boldsymbol{x}) \in \Pi$ then $\forall \mathbf{id}\ M(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x}) \mapsto \mathsf{yes}$, whereas if $(G, \boldsymbol{s}, \boldsymbol{x}) \notin \Pi$ then $\forall \mathbf{id}\ M(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x}) \mapsto \mathsf{no}$.*

By definition, $\mathsf{MAD_s}$ is a subset of both MAD and co-MAD and it is easy to check that $\mathsf{MAD_s} = \mathsf{co\text{-}MAD_s}$. Moreover, all of these classes are subsets of Δ_1^0, since a centralized algorithm, provided with an encoding of the graph and the starting positions, inputs, and IDs of the agents, can simulate the corresponding mobile agent protocol and decide appropriately. As mentioned in [13], path is an example of a mobile agent decision problem which is in $\Delta_1^0 \setminus \mathsf{MAD}$, since, intuitively, an agent cannot distinguish a long path from a cycle. In fact, this observation yields $\mathsf{path} \in \Delta_1^0 \setminus (\mathsf{MAD} \cup \mathsf{co\text{-}MAD})$.

A nontrivial problem in $\mathsf{MAD_s}$ is treesize. The problem was already shown to be in MAD in [13]. For the stronger property that $\mathsf{treesize} \in \mathsf{MAD_s}$, we need a modification of the protocol given in [13].

Proposition 1. $\mathsf{treesize} \in \mathsf{MAD_s}$.

We now show that MAD and co-MAD are strict supersets of $\mathsf{MAD_s}$.

Proposition 2. $\mathsf{allempty} \in \mathsf{MAD} \setminus \mathsf{MAD_s}$ *and* $\overline{\mathsf{allempty}} \in \mathsf{co\text{-}MAD} \setminus \mathsf{MAD_s}$.

As we mentioned, $\mathsf{MAD_s}$ is included in both MAD and co-MAD. In fact, $\mathsf{MAD_s} = \mathsf{MAD} \cap \mathsf{co\text{-}MAD}$. We state this as a theorem without proof, since it can be obtained as a corollary of Theorems 2 and 3, which we will prove in Section 5.

Theorem 1. $\mathsf{MAD_s} = \mathsf{MAD} \cap \mathsf{co\text{-}MAD}$.

By Theorem 1, if allempty was included in co-MAD, we would obtain $\mathsf{allempty} \in \mathsf{MAD_s}$, which we know to be false. Thus, $\mathsf{allempty} \notin \mathsf{co\text{-}MAD}$ and we obtain a separation between MAD and co-MAD. Symmetrically, $\overline{\mathsf{allempty}} \in \mathsf{co\text{-}MAD} \setminus \mathsf{MAD}$.

4 Interleaving Multiple Mobile Agent Protocols

It is important to have a tool that enables the execution of several mobile agent protocols on the same instance, and that also permits the mobile agents to make decisions based on the outcomes of these executions. For example, if one were to give a direct proof of Theorem 1 above, one would need a way for the agents to coordinate in order to execute both the MAD and the co-MAD protocol for a particular problem, and then, based on the outcome of these executions, to give a unanimous correct answer (in the spirit of $\mathsf{MAD_s}$).

In classical computing, the well known *dovetailing* technique achieves this interleaving of different computations. Classical dovetailing proceeds in phases: in phase T, the first T steps of the first T programs are executed. At this point, an auxiliary function is executed, which decides, based on these executions, whether to accept, reject, or continue with the next phase. Correspondingly, the mobile agent meta-protocol which we propose in this section, proceeds in phases: in phase T, the agents execute the first T steps of the first T mobile agent protocols and then decide whether to accept, reject, or proceed to the next phase. In the mobile agent case, each agent decides independently by locally executing

a function, which is given as a parameter to the meta-protocol. We call this function a *local decider*.

Still, it may happen that one or more agents halt as a result of executing the local decider, while others decide to continue. In such a case, the execution of the protocols in the next phase could be corrupted because the halted agents no longer follow the protocol. However, these halted agents can now be regarded as fixed tokens and the meta-protocol uses them in order to create a map of the graph. In fact, this is done in such a way as to ensure that all non-halted agents obtain not only the map of the graph but actually full knowledge of the implicit input. Based on this knowledge, each agent decides irrevocably whether to accept or reject by means of a second function which is given as a parameter to the meta-protocol, and which we call a *global decider*.

4.1 Ingredients of the meta-protocol

We propose a generic meta-protocol $\mathcal{P}_{\mathcal{N},f,g}$, which is parameterized by $\mathcal{N}, f, g$. The set $\mathcal{N}$ is a, possibly infinite, recursively enumerable set of mobile agent protocols. Let N_i, $i \geq 0$, denote the i-th protocol in such an enumeration. The functions f and g are computable functions which represent local computations with the following specifications:

Global decider: The function f maps pairs consisting of an explicit and an implicit input, i.e., tuples of the form $(\mathsf{id}, x; \mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$, to the set $\{\mathbf{accept}, \mathbf{reject}\}$. In this case, we say that f is a *global decider*. When an agent executes f, it halts by accepting or rejecting according to the outcome of f.

Local decider: The function g takes as input an explicit input (id, x) and a list $(H_1, \ldots, H_\sigma)$ of arbitrary length σ, where each H_j is the history of the partial execution of $N_j(\mathsf{id}, x; \mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$ for a certain number of steps and $(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$ is an implicit input common for all histories $H_1, \ldots, H_\sigma$. The outcome of g is one of $\{\mathbf{accept}, \mathbf{reject}, \mathbf{continue}\}$. When an agent executes g, it halts in the corresponding state if the outcome is **accept** or **reject**, otherwise it continues without halting.

If for every implicit input $(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$ and for every T_0, there exists a $T \geq T_0$ and some i such that the local computation $g(\mathsf{id}_i, x_i, H_1, \ldots, H_{\min(T,|\mathcal{N}|)})$ returns either **accept** or **reject**, where each H_j is an encoding of the execution of $N_j(\mathsf{id}_i, x_i; \mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$ for T steps, then we say that g is a *local decider for* $\mathcal{N}$.

The meta-protocol uses the following procedures CREATE-MAP and RDV:

Procedure CREATE-MAP(R): An agent executes this procedure only when it is on a node which contains at least one halted (or idle) agent. Starting from this node, and treating the halted agent as a fixed mark, it attempts to create a map of the graph assuming that the graph contains at most R nodes. More precisely, the agent first creates a map consisting in a single node corresponding to the marked node r, with d_r pending edges with port numbers from 1 to d_r. Then, while there remain some pending edges and there are at most R explored nodes, the agent explores some arbitrary pending edge as follows. The agent goes to the known extremity u of the pending edge by using the map and traverses it.

It then determines whether its current position v corresponds to a node of its map, as follows: For every node w in its map, it computes a path in the map going from w to r and follows the corresponding sequence of port numbers in the unknown graph, starting from v. If it leads to the marked node, then $v = w$ and the agent updates its map by linking the pending edges of u and w with the appropriate port numbers. Otherwise, it retraces its steps to come back to v and tests a next node w. If all nodes turn out to be different from v, then the agent goes back to the marked node through u, and updates its map by adding a new node corresponding to v, linked to u, and with the appropriate number of new pending edges. At the end of the procedure, the agent either has a complete map of the graph, or knows that the graph has more than R nodes. This procedure takes at most $4R^4$ steps.

Procedure RDV(R, id): This procedure guarantees that a group of k agents which (a) know the same upper bound R on the number of nodes in the graph, (b) have distinct id's $\{\mathsf{id}_1, \ldots, \mathsf{id}_k\}$, and (c) start executing RDV(R, id_i) at the same time from different nodes s_i, will all meet each other after finite time. Moreover, each agent knows when it has met all other agents executing RDV, even without initial knowledge of k.

The RDV procedure uses as a subroutine the following EXPLORE-BALL procedure: An agent executing EXPLORE-BALL(R) attempts to explore the ball of radius R around its starting node s_i, assuming an upper bound of R on the maximum degree of the graph. This is achieved by having the agent try every sequence of length R of port numbers from the set $\{1, \ldots, R\}$, retracing its steps backward after each sequence to return to s_i. If a particular sequence instructs the agent to follow a port number that does not exist at the current node (i.e., the port number is larger than the degree of the node), then the agent aborts that sequence and returns to s_i. Attempting all possible sequences takes at most $B(R) = 2R \cdot R^R$ steps. If an agent finishes earlier, it waits on s_i until $B(R)$ steps are completed. Therefore, a team of agents that start executing EXPLORE-BALL(R) at the same time from different nodes are synchronized and back at their starting positions after $B(R)$ steps.

Now, for each bit of id_i, the RDV procedure executes the following: If the bit is 0, the agent waits at s_i for $B(R)$ steps and then executes EXPLORE-BALL(R), whereas if the bit is 1, the agent first executes EXPLORE-BALL(R) and then waits on its starting position for $B(R)$ steps. After it exhausts the bits of id_i, the agent executes twice EXPLORE-BALL(R). This guarantees that, if the number of nodes is at most R, then after $2 \cdot (|\mathsf{id}_i| + 1) \cdot B(R)$ steps, each agent i is located at s_i and has met all other agents executing RDV. Note that after every integer multiple of $B(R)$ steps, each agent is located at its initial node s_i.

4.2 Description of the meta-protocol

The meta-protocol $\mathcal{P}_{\mathcal{N},f,g}$ works in phases, which correspond to increasing values of a presumed upper bound T on the number of nodes in the graph, the length of all agent identifiers, and the completion time of protocols $N_1, \ldots, N_T$. We will

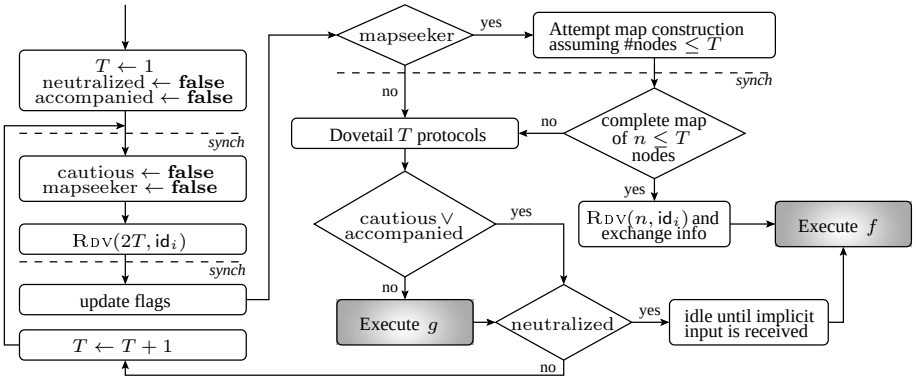

Fig. 2. High-level flowchart of the meta-protocol of Section 4.

say that an agent is *idle* if it is waiting indefinitely on its starting node for some other agent to provide it with the knowledge of the full implicit input. We will say that an agent is *participating* if it is not halted and not idle. Note that an agent may halt only as a result of executing one of the decider functions f and g. In each phase T, the agents perform the following actions (see also Fig. 2):

Search for nearby starting positions and set flags. Each participating agent i first executes $\textsc{Rdv}(2T, \mathsf{id}_i)$ for at most $2(T+1)B(2T)$ steps. By design of Rdv, this guarantees that agent i will explore its $2T$-neighborhood at least once and, in particular, if $T \geq |\mathsf{id}_i|$, then for each other participating agent, agent i will explore its $2T$-neighborhood at least once with that agent staying on its starting node. If, in the process, the agent meets any agent, then it sets its accompanied flag. It also sets its neutralized flag if the encountered agent is participating and it has a lexicographically larger ID. If the encountered agent is halted or idle, the agent sets its mapseeker flag. Finally, if the agent finds a node with degree larger than $2T$ or if the length of its ID is greater than T, it sets its cautious flag. All agents synchronize at this point.

Mapseeker agents attempt to create a map of the graph. Next, each agent i with the mapseeker flag set moves to a halted or idle agent which it has found previously, while executing Rdv in the current phase. Then, it attempts to create a map of the graph by executing $\textsc{Create-Map}(T)$ and returns to s_i. Overall, this takes at most $4T^4 + 4T$ steps. Moreover, during the execution of Create-Map, mapseeker agents collect starting position and input information from all halted and idle agents that they encounter. Meanwhile, non-mapseeker agents wait for $4T^4 + 4T$ steps. All agents synchronize at this point.

So far, we have achieved that, if $T \geq n$, where n is the number of nodes in G, then either no agent is a mapseeker having the full map of G, or all participating agents have the mapseeker flag set and they have the full map of G (Lemma 1 below). If all mapseeker agents have the full map of G and $T \geq n$, then each such agent i executes $\textsc{Rdv}(n, \mathsf{id}_i)$, which guarantees that, finally, it is located at s_i and has met all other agents executing Rdv. Therefore, after concluding the

RDV procedure, each mapseeker executes f with full knowledge of the implicit input (Lemma 2).

Perform dovetailing. At this point, if no agent is a mapseeker having the full map of G, the agents execute each of the protocols $N_1, \ldots, N_{\min(T,|\mathcal{N}|)}$ for at most T steps, and then retrace backward to s_i (agents are synchronized after executing each protocol). If any of these protocols instructs an agent to halt, the agent instead waits until the T-step execution period has finished, and then returns to s_i. If the agent does not have the cautious or accompanied flags set, it then executes $g(\mathsf{id}, x, H_1, \ldots, H_{\min(T,|\mathcal{N}|)})$, where H_j is the history of the T-step execution of N_j with explicit input (id, x). Since this process takes at most $2T^2$ steps, all agents that do not halt as a result of executing g are synchronized at the end of the current phase. It is guaranteed that the histories fed to the local decider g correspond to correct executions of the corresponding protocols for implicit input $(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$, even though some of the agents may have halted or become idle in earlier phases (Lemma 3 and Corollary 1).

Neutralized agents become idle. Finally, at the end of the phase, neutralized agents start waiting for the implicit input (i.e., they become idle), and when they receive it (from some mapseeker agent), they execute the global decider f.

Lemma 1. *In each phase, either all or none of the participating agents (i.e., non-halted and non-idle) execute f.*

Lemma 2. *Any agent that executes f has full knowledge of the implicit input $(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$.*

Lemma 3. *If an agent i executes g during phase T, then no other agent's starting node is at distance $2T$ or less from s_i.*

By Lemma 3, we obtain following corollary:

Corollary 1. *Any agent i that executes g has histories which correspond to the correct histories of $N_j(\mathsf{id}_i, x_i; \mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$ for T steps ($1 \leq j \leq \min(T, |\mathcal{N}|)$), even though some of the agents may have halted or become idle in earlier phases.*

In view of Corollary 1, we can show that all agents terminate and, in fact, they all terminate on their respective starting nodes.

Lemma 4. *Let f be a global decider and let g be a local decider for $\mathcal{N}$. Then, each agent halts under the execution $\mathcal{P}_{\mathcal{N},f,g}(\mathbf{id}, G, \boldsymbol{s}, \boldsymbol{x})$ by executing either f or g. Moreover, each agent i halts on its starting node s_i.*

4.3 Application of the meta-protocol

To summarize, the meta-protocol is a generic tool that enables us to interleave the executions of a possibly infinite set of mobile agent protocols. Eventually, each agent accepts or rejects, based either on the histories of the executions of a

number of these protocols (by means of the local decider), or on full knowledge of the implicit input (by means of the global decider).

We use the meta-protocol in order to place a particular problem in one of the mobile agent computability classes of Table 1. A common part of the proofs consists in defining the list of protocols $\mathcal{N}$ and suitable deciders f and g, and in showing that f and g indeed satisfy the global and local decider properties, respectively. This is followed by a part tailored to each particular result, where we use the properties of the meta-protocol (Lemmas 1–4 and Corollary 1) and the particular definitions of f and g, in order to show that agents that execute $\mathcal{P}_{\mathcal{N},f,g}$ always terminate in the desired state. The desired state is indicated by the class in which we wish to place the problem. For example, if we wish to show that a problem is in $\mathsf{MAD_s}$, we will have to show that all agents give the correct answer for all implicit inputs.

5 Mobile Agent Verifiability Classes

Definition 5. *A decision problem Π is in $\mathsf{MAV_s}$ if and only if there exists a protocol M such that for all instances $(G, \boldsymbol{s}, \boldsymbol{x})$: if $(G, \boldsymbol{s}, \boldsymbol{x}) \in \Pi$ then $\exists \boldsymbol{y}\ \forall \mathbf{id}\ M(\mathbf{id}, G, \boldsymbol{s}, \langle \boldsymbol{x}, \boldsymbol{y} \rangle) \mapsto \mathsf{yes}$, whereas if $(G, \boldsymbol{s}, \boldsymbol{x}) \notin \Pi$ then $\forall \boldsymbol{y}\ \forall \mathbf{id}\ M(\mathbf{id}, G, \boldsymbol{s}, \langle \boldsymbol{x}, \boldsymbol{y} \rangle) \mapsto \mathsf{no}$.*

By definition, $\mathsf{MAV_s} \subseteq \mathsf{MAV}$. Moreover, $\mathsf{MAV} \subseteq \Sigma_1^0$, since a centralized algorithm can simulate the MAV protocol for all possible certificate vectors (by classical dovetailing) and accept if it finds a certificate for which all agents accept. By taking complements, we obtain as well that $\mathsf{co\text{-}MAV_s} \subseteq \mathsf{co\text{-}MAV} \subseteq \Pi_1^0$.

There exist several nontrivial problems in $\mathsf{MAV_s}$ and $\mathsf{co\text{-}MAV_s}$ (Proposition 3). Furthermore, we can show that MAV is a strict superset of $\mathsf{MAV_s}$ and, as a corollary, co-MAV is a strict superset of $\mathsf{co\text{-}MAV_s}$ (Proposition 4).

Proposition 3. *For any fixed $\gamma \geq 1$, $\mathsf{degree}_\gamma \in \mathsf{MAV_s}$. Furthermore, $\mathsf{consensus} \in \mathsf{co\text{-}MAV_s}$ and $\mathsf{alone} \in \mathsf{co\text{-}MAV_s}$.*

Proposition 4. $\mathsf{degree} \in \mathsf{MAV} \setminus (\mathsf{MAV_s} \cup \mathsf{co\text{-}MAV})$.

Proposition 4 also separates MAV from co-MAV. In order to separate Σ_1^0 from MAV and Π_1^0 from co-MAV, we observe that the teamsize problem, which is clearly in $\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$, is neither in MAV nor in co-MAV.

Proposition 5. $\mathsf{teamsize} \in \Delta_1^0 \setminus (\mathsf{MAV} \cup \mathsf{co\text{-}MAV})$.

Decision problems with "no" certificates In classical computability, the class $\Pi_1^0 = \mathsf{co}\text{-}\Sigma_1^0$ can be seen as the class of problems that admit a "no" certificate, i.e.: for "no" instances, there exists a certificate that leads to rejection, whereas for "yes" instances, no certificate can lead to rejection. In this respect, while MAV can certainly be considered as the mobile agent analogue of Σ_1^0, co-MAV is not quite the analogue of Π_1^0. Problems in co-MAV indeed admit a "no" certificate, but the acceptance mechanism is reversed: for "no" instances,

there exists a certificate that leads all agents to reject. This motivates us to define and study co-MAV′, the class of mobile agent problems that admit a "no" certificate while retaining the MAV acceptance mechanism, as well as its complement MAV′. We give the definition of MAV′ below.

Definition 6. *A decision problem Π is in* MAV′ *if and only if there exists a protocol M such that for all instances $(G, \boldsymbol{s}, \boldsymbol{x})$: if $(G, \boldsymbol{s}, \boldsymbol{x}) \in \Pi$ then $\exists \boldsymbol{y}\ \forall \mathbf{id}\ M(\mathbf{id}, G, \boldsymbol{s}, \langle \boldsymbol{x}, \boldsymbol{y} \rangle) \mapsto \widetilde{\mathsf{yes}}$, whereas if $(G, \boldsymbol{s}, \boldsymbol{x}) \notin \Pi$ then $\forall \boldsymbol{y}\ \forall \mathbf{id}\ M(\mathbf{id}, G, \boldsymbol{s}, \langle \boldsymbol{x}, \boldsymbol{y} \rangle) \mapsto \mathsf{no}$.*

By definition, it holds that $\mathsf{MAV_s} \subseteq \mathsf{MAV}'$ and $\mathsf{co\text{-}MAV_s} \subseteq \mathsf{co\text{-}MAV}'$. To show $\mathsf{MAV}' = \mathsf{MAV_s}$ (and thus $\mathsf{co\text{-}MAV}' = \mathsf{co\text{-}MAV_s}$), we need to "boost" the MAV′ protocol so that the agents answer unanimously even in "yes" instances. We achieve this by supplying an extra certificate, which is interpreted as the number of nodes of the graph. This enables the agents to meet and exchange information in "yes" instances, and therefore reach a unanimous decision. The meta-protocol from Section 4 essentially provides "for free" the necessary subroutines for meeting and exchanging information.

Theorem 2. $\mathsf{MAV}' = \mathsf{MAV_s}$ *and* $\mathsf{co\text{-}MAV}' = \mathsf{co\text{-}MAV_s}$.

In view of Theorem 2, it follows that $\mathsf{MAV_s} \subseteq \mathsf{MAV} \cap \mathsf{co\text{-}MAV}$ and $\mathsf{co\text{-}MAV_s} \subseteq \mathsf{MAV} \cap \mathsf{co\text{-}MAV}$. We separate $\mathsf{MAV} \cap \mathsf{co\text{-}MAV}$ from both of these classes with the problem mineven:

Proposition 6. $\mathsf{mineven} \in (\mathsf{MAV} \cap \mathsf{co\text{-}MAV}) \setminus (\mathsf{MAV_s} \cup \mathsf{co\text{-}MAV_s})$.

Connections with the decidability classes We explore the relationships among the decidability classes of Section 3 and the classes defined in this section. From the definitions we know that $\mathsf{MAD} \subseteq \mathsf{co\text{-}MAV}'$, therefore, by Theorem 2, $\mathsf{MAD} \subseteq \mathsf{co\text{-}MAV_s}$. Similarly, $\mathsf{co\text{-}MAD} \subseteq \mathsf{MAV_s}$. Therefore, since $\mathsf{MAD_s} \subseteq \mathsf{MAD} \cap \mathsf{co\text{-}MAD}$, we also have that $\mathsf{MAD_s} \subseteq \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV_s}$.

We show in Theorem 3 that, in fact, $\mathsf{MAD_s} = \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV_s}$. Furthermore, from the definitions and Theorem 2, we have $\mathsf{MAD} \subseteq \mathsf{MAV} \cap \mathsf{co\text{-}MAV_s}$ and $\mathsf{co\text{-}MAD} \subseteq \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV}$. We show that these actually hold as equalities in Theorem 4 below. The proof of Theorem 3 (resp. Theorem 4) is based on trying all possible combinations of certificates for the $\mathsf{MAV_s}$ (resp. MAV) and $\mathsf{co\text{-}MAV_s}$ protocols. Here, we use the full power of the meta-protocol of Section 4 in order to interleave and synchronize this infinite number of executions.

Theorem 3. $\mathsf{MAD_s} = \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV_s}$.

Theorem 4. $\mathsf{MAD} = \mathsf{MAV} \cap \mathsf{co\text{-}MAV_s}$ *and* $\mathsf{co\text{-}MAD} = \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV}$.

Note that it was shown in [13] that, if we consider decision problems that are decidable or verifiable by a single agent (thus giving rise to the classes $\mathsf{MAD_1}$ and $\mathsf{MAV_1}$), then it holds that $\mathsf{MAD_1} = \mathsf{MAV_1} \cap \mathsf{co\text{-}MAV_1}$. Theorems 3 and 4 can be seen as generalizations of that result to multiagent classes.

Proposition 7. *For any fixed* $\gamma \geq 1$, $\mathsf{degree}_\gamma \in \mathsf{MAV_s} \setminus \mathsf{co\text{-}MAD}$ *and* $\overline{\mathsf{degree}_\gamma} \in \mathsf{co\text{-}MAV_s} \setminus \mathsf{MAD}$.

In view of Theorem 4, Proposition 7 yields a separation between $\mathsf{MAV_s}$ and co-MAV, as $\mathsf{degree}_\gamma \in \mathsf{MAV_s} \setminus \mathsf{co\text{-}MAV}$, and a separation between $\mathsf{co\text{-}MAV_s}$ and MAV, as $\overline{\mathsf{degree}_\gamma} \in \mathsf{co\text{-}MAV_s} \setminus \mathsf{MAV}$.

By combining the results of this section with the results of Section 3, we obtain a picture of the relationships among the classes below MAV and co-MAV, as illustrated in Figure 1.

References

1. Boldi, P., Vigna, S.: An effective characterization of computability in anonymous networks. In: DISC 2001. LNCS, vol. 2180, pp. 33–47. Springer (2001)
2. Boldi, P., Vigna, S.: Universal dynamic synchronous self-stabilization. Distrib. Comput. 15(3), 137–153 (2002)
3. Chalopin, J., Godard, E., Métivier, Y.: Local terminations and distributed computability in anonymous networks. In: DISC 2008. LNCS, vol. 5218, pp. 47–62. Springer (2008)
4. Chalopin, J., Godard, E., Métivier, Y., Tel, G.: About the termination detection in the asynchronous message passing model. In: SOFSEM 2007. LNCS, vol. 4362, pp. 200–211. Springer (2007)
5. Chandra, T.D., Hadzilacos, V., Toueg, S.: The weakest failure detector for solving consensus. J. ACM 43(4), 685–722 (1996)
6. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM 43(2), 225–267 (1996)
7. Das, S.: Mobile agents in distributed computing: Network exploration. Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 109, 54–69 (2013)
8. Das S., Kutten S., Lotker Z.: Distributed verification using mobile agents. In: ICDCN 2013. LNCS, vol 7730, pp. 330–347. Springer (2013)
9. Fraigniaud, P., Göös, M., Korman, A., Parter, M., Peleg, D.: Randomized distributed decision. Distrib. Comput. 27(6), 419–434 (2014)
10. Fraigniaud, P., Göös, M., Korman, A., Suomela, J.: What can be decided locally without identifiers? In: PODC 2013. pp. 157–165. ACM (2013)
11. Fraigniaud, P., Halldórsson, M.M., Korman, A.: On the impact of identifiers on local decision. In: OPODIS 2012. LNCS, vol. 7702, pp. 224–238. Springer (2012)
12. Fraigniaud, P., Korman, A., Peleg, D.: Towards a complexity theory for local distributed computing. J. ACM 60(5), 35 (2013)
13. Fraigniaud, P., Pelc, A.: Decidability classes for mobile agents computing. In: LATIN 2012. LNCS, vol. 7256, pp. 362–374. Springer (2012)
14. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. 13(1), 124–149 (1991)
15. Lange, D.B., Oshima, M.: Seven good reasons for mobile agents. Commun. ACM 42(3), 88–89 (1999)
16. Markou, E.: Identifying hostile nodes in networks using mobile agents. Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 108, 93–129 (2012)
17. Yamashita, M., Kameda, T.: Computing functions on asynchronous anonymous networks. Math. Syst. Theory 29(4), 331–356 (1996)