On Mobile Agent Verifiable Problems[☆]

Evangelos Bampas[a], David Ilcinkas[b]

[a]*LIF, Aix-Marseille University and CNRS, Marseille, France*
[b]*LaBRI, CNRS and Univ. Bordeaux, France*

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 [J. Parallel Distrib. Comput, vol. 109, pp. 117–128], we introduce several natural new computability classes allowing for a finer classification of problems below MAV or its complement class co-MAV, the former 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 (introduced in [8]), as well as wait-free hierarchies (introduced in [22]), which both concern crash-fault-tolerance in distributed asynchronous

[☆]A preliminary version of this work appears in the Proceedings of the 12th Latin American Theoretical Informatics Symposium, LNCS vol. 9644, pp.123–137, Springer, 2016.

Email addresses: `evangelos.bampas@lif.univ-mrs.fr` (Evangelos Bampas), `david.ilcinkas@labri.fr` (David Ilcinkas)

systems. An unreliable failure detector is an external failure detection mechanism that can make mistakes. It is composed of local modules, one on each node, which output a set of processes that the failure detector module suspects have crashed. Chandra and Toueg [8] introduced this notion and a way to compare unreliable failure detectors, and they exhibited and studied an infinite hierarchy of failure detector classes, leading to a way of classifying distributed tasks, according to the weakest failure detector allowing the given task to be solved. Another approach consists in directly classifying concurrent objects, which are data structures shared by concurrent processes. This line of work, inspired by the seminal paper [22], considers wait-free implementations. These are implementations such that any operation on the object by a process terminates in a finite number of steps, even if other processes crash or have different progress speeds. A so-called wait-free hierarchy of objects is then constructed by classifying objects depending on whether an object has a wait-free implementation only using instances of another object as communication primitives. Finally, a more recent work [20] deals with checkability of distributed tasks: the decision problem associated to a task consists in determining whether a given output is valid with respect to the task specification.

Computability- and complexity-theoretic studies for decision problems also concern the fault-free distributed $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$ models. In both models, a communication graph describes which nodes are able to directly communicate. The nodes have unique IDs, and they operate in synchronous rounds, in which any node is able to send a (possibly different) message to each of its neighbor. There are no restrictions on the memory or computing capabilities of the nodes. In the $\mathcal{LOCAL}$ model, there are even no restrictions on the size of the messages, while the $\mathcal{CONGEST}$ model usually assumes that each message has size at most $O(\log n)$ bits.

In both models, several papers studied different classes of distributed languages. A distributed language is basically a set of labeled networks. For example, the set of properly colored networks is a distributed language. In particular, decision and verification were studied. A distributed language is decidable if there exists a distributed algorithm able to globally determine whether the input labeled network is in the language, while a distributed language is verifiable if the membership of an instance to the language can be checked by a distributed algorithm with the help of certificates, in a similar manner as certificates are used in the centralized complexity class NP. System-wide acceptance is usually defined as all nodes locally accepting. System-wide rejection is usually defined as at least one node rejecting.

Deterministic and randomized decision classes in the $\mathcal{LOCAL}$ model were introduced in [26] and [18]. The latter paper also introduced one verification class, which is somehow a variant of another verification class introduced by Korman, Kutten, and Peleg [23]. The impact of identifiers or the lack of them has also been investigated [13, 16, 17]. In the $\mathcal{CONGEST}$ model, decision and verification were also considered [11, 21], as well as a distributed version of property testing [5]. For a much more detailed survey, see [14].

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 [3, 4, 6, 7, 28]. Finally, recent works focus on the computational power of teams of mobile agents [10, 19]. Our work lies in this latter direction.

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 [24], and has been of significant interest to the distributed computing community (see, e.g., the surveys on graph exploration [9], identification of hostile nodes [25], or rendezvous [27]). 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.

One may argue about the usefulness of developing a theory specifically for mobile agent decision problems, apart from its inherent theoretical interest. There are several reasons.

On the one hand, we believe that 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 [9], rendezvous [27], graph searching [15], etc., which are not decision problems. 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.

On the other hand, we think that decision problems are inherently interesting, despite the relative shortage of studies devoted to them ([10, 19]). Most studies so far in mobile agent computing indeed concern "complex" problems, in the sense that either the output is not binary (constructing the map of the network, counting the number of agents or nodes, etc.) or the problem requires specific terminal configurations (like in rendezvous) or even properties on the sequences of configurations (like in exploration or graph searching). Decision problems are however closely related to these more complex problems. First, a significant proportion of the studies make initial assumptions on the maximum and/or minimum number of agents in the network [2, 12], on the topology (like assuming that the agents are on a tree [1]), or more generally on the possible initial configurations. Algorithms solving decision problems can be used to check that such assumptions actually hold, before running the algorithm dedicated to solve the problem at hand, which may consume resources. Second, certain contexts are subject to faults. In such cases, algorithms solving decision problems may be used for fault tolerance purposes: agents can check (possibly by means of certificates that were constructed while solving the main problem) whether the achieved configuration or output satisfies the desired properties.

In this paper, we consider one of the most broadly used models of mobile agent computing, the same as the one studied in [19]. More precisely, we consider a distributed system in which computation is performed by one or more deterministic mobile agents, operating in an unknown, anonymous network (nodes have no identifiers and edges are only locally distinguished). Each agent is modeled as a deterministic Turing machine, 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.

1.2. Related work

In [19], Fraigniaud and Pelc introduced two natural computability classes, MAD and MAV, as well as their counterparts $\mathsf{co\text{-}MAD}$ and $\mathsf{co\text{-}MAV}$. 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 [19] 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 [10], 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 [10], the authors study the minimal amount of memory needed by the mobile agents to distributedly verify some classes of graph properties. Again, the studied properties are different from the ones studied here and in [19], 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 $\mathsf{co\text{-}MAV}$. The classes $\mathsf{MAD_s}$ and $\mathsf{MAV_s}$ are strict versions of MAD and MAV, respectively, in which unanimity is required in both "yes" and "no" instances. Furthermore, we consider the class $\mathsf{co\text{-}MAV'}$ (and its counterpart $\mathsf{MAV'}$) of mobile agent decision problems that admit a certificate for "no" instances, while retaining the system-wide acceptance mechanism of MAV. The inclusion diagram that contains the inclusions known by definition of these classes (i.e., prior to this work) is shown in Figure 1.

We perform a thorough investigation of the relationships between the newly introduced and pre-existing classes (Sections 3 and 5). As a result, we obtain a complete Venn diagram (Figure 2) which illustrates the tight interconnections

$$\begin{tikzcd}
\text{co-MAV}' & & \text{co-MAV} & & \text{MAV} & & \text{MAV}' \\
& \text{co-MAV}_s \arrow[ul] \arrow[ur] & & & & \text{MAV}_s \arrow[ul] \arrow[ur] & \\
& \text{MAD} \arrow[uul] \arrow[uurrr] & & & & \text{co-MAD} \arrow[uulll] \arrow[uur] & \\
& & & \text{MAD}_s \arrow[ull] \arrow[urr] & & &
\end{tikzcd}$$

Figure 1: An inclusion diagram that illustrates only the inclusions that are known by definition of the classes considered in this paper. This represents our knowledge about these classes prior to this work. Class definitions are summarized in Table 1.

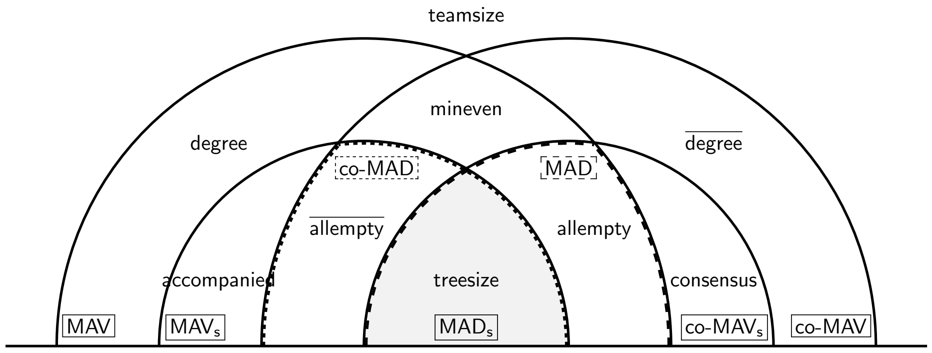


Figure 2: Containments between classes below MAV and co-MAV with corresponding illustrative problems. Class and problem definitions are summarized in Tables 1 and 2, respectively.

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: $\mathsf{MAD} \subseteq \mathsf{co\text{-}MAV}$ and $\mathsf{co\text{-}MAD} \subseteq \mathsf{MAV}$. Figure 3 contains the inclusion diagram of the considered classes that we obtain as a result of this work.

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 (Section 6). We conclude with a discussion on a decidability class with a much less strict acceptance mechanism, in which only one among the participating agents is required to give the correct answer (Section 7). 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 (Section 4). This can be seen as a mobile agent computing analogue of the well-known dovetailing technique from classical recursion theory.

$$\begin{tikzcd}
 & \mathsf{co\text{-}MAV} & & \mathsf{MAV} & \\
 & & \mathsf{MAV} \cap \mathsf{co\text{-}MAV} \arrow[ul] \arrow[ur] & & \\
 & \mathsf{co\text{-}MAV'} = \mathsf{co\text{-}MAV_s} \arrow[u] & \mathsf{MAD} \cup \mathsf{co\text{-}MAD} \arrow[u] & \mathsf{MAV_s} = \mathsf{MAV'} \arrow[u] & \\
 & \mathsf{MAV} \cap \mathsf{co\text{-}MAV_s} = \mathsf{MAD} \arrow[u] \arrow[ur] & & \mathsf{co\text{-}MAD} = \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV} \arrow[u] \arrow[ul] & \\
 & & \begin{array}{c} \mathsf{MAD_s} \\ = \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV_s} \\ = \mathsf{MAD} \cap \mathsf{co\text{-}MAD} \end{array} \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

Figure 3: The inclusion diagram of the classes below MAV and co-MAV that illustrates the results obtained in this work. Class definitions are summarized in Table 1.

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, $\widehat{\text{yes}}$ (resp. $\widehat{\text{no}}$) 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:) $\widehat{\text{no}}$	✗	✓	✗
co-MAD	($\forall$ certificate:) $\widehat{\text{yes}}$	($\forall$ certificate:) no	✓	✗	✗
$\mathsf{MAV_s}$	$\exists$ certificate: yes	$\forall$ certificate: no	✓	✓	✗
co-$\mathsf{MAV_s}$	$\forall$ certificate: yes	$\exists$ certificate: no	✓	✓	✗
MAV	$\exists$ certificate: yes	$\forall$ certificate: $\widehat{\text{no}}$	✗	✓	✗
co-MAV	$\forall$ certificate: $\widehat{\text{yes}}$	$\exists$ certificate: no	✓	✗	✗
MAV′	$\exists$ certificate: $\widehat{\text{yes}}$	$\forall$ certificate: no	✓	✓	✗
co-MAV′	$\forall$ certificate: yes	$\exists$ certificate: $\widehat{\text{no}}$	✓	✓	✗

2. Preliminaries

The graphs in which the mobile agents operate are simple, 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. If v is a node, $N(v)$ stands for the set of neighboring nodes of v. A port-labeled graph G is a triple $G = \big(V, E, (\lambda_v)_{v \in V}\big)$, where $\lambda_v : N(v) \to \{1, \ldots, d_v\}$ is a function such that $\lambda_v(u)$ is the local port number at v that leads to its neighbor u. An *automorphism of G that preserves port numbers* is a function $\alpha : V \to V$ such that, for all $u, v \in V$, $\{u, v\} \in E \Leftrightarrow \big\{\alpha(u), \alpha(v)\big\} \in E$ and, additionally, $\lambda_v(u) = \lambda_{\alpha(v)}\big(\alpha(u)\big)$.

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 $\vec{x}$ is a list, then $|\vec{x}|$ is the length of $\vec{x}$ and x_i is the i-th element of $\vec{x}$. We will use a bold typeface for lists and a light typeface for list elements.[1] If f is a function that can be applied to the elements of $\vec{x}$, then we will use the notation $f(\vec{x}) = \big(f(x_1), \ldots, f(x_{|\vec{x}|})\big)$. In the same spirit, if $\vec{x}$ and $\vec{y}$ are equal-length lists of strings, then $\langle\vec{x}, \vec{y}\rangle$ stands for the list $\big(\langle x_1, y_1\rangle, \ldots, \langle x_{|\vec{x}|}, y_{|\vec{y}|}\rangle\big)$.

We recall the standard notation $\mathsf{\Sigma}^0_j$ for the j-th level of the arithmetic hierarchy ($j \geq 1$), as well as $\mathsf{\Pi}^0_j = \mathsf{co}\text{-}\mathsf{\Sigma}^0_j$ and $\mathsf{\Delta}^0_j = \mathsf{\Sigma}^0_j \cap \mathsf{\Pi}^0_j$. In particular, $\mathsf{\Sigma}^0_1$ is the set of recursively enumerable (or Turing-acceptable) languages and $\mathsf{\Delta}^0_1$ is the set of Turing-decidable languages. If $\mathsf{HP}_0 = \emptyset$, then the problem $\mathsf{HP}_j = \{x \in \Sigma^\star : x \text{ is the encoding of a TM with oracle } \mathsf{HP}_{j-1} \text{ that halts with input } \varepsilon\}$ is known to be $\mathsf{\Sigma}^0_j$-complete under the many-one reduction.

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, $\vec{\mathsf{id}}$ be a list of distinct IDs, $\vec{s}$ be a list of nodes of G, and $\vec{x}$ be a list of strings such that $|\vec{\mathsf{id}}| = |\vec{s}| = |\vec{x}| = k > 0$. We denote by $M(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ 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 $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ is called the *implicit input.* Similarly, we denote by $M(\mathsf{id}, x; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ 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.

[1] In this accepted manuscript, we use the arrow (vector) notation for lists.

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

Definition 1 (View). *The* view $\mathcal{V}_G(u)$ *from a node u in a graph G with local port numbers is a possibly infinite tree (W, F) with local port numbers, defined as follows: The node set W of the tree contains all finite paths starting from u in G, except those that traverse back and forth the same edge in consecutive steps. There is an edge $\{w_1, w_2\} \in F$ if and only if w_1 corresponds to some path $u \rightsquigarrow p$ in G and w_2 corresponds to the same path augmented by some edge $p \rightarrow q$. The port numbers at the endpoints of $\{w_1, w_2\}$ are the same as those at the respective endpoints of $\{p, q\}$ in G. We refer to the node of the view that corresponds to the unique zero-length path from u in G as the* root *of the view.*

Definition 2 (Truncated view). *For $T \geq 0$, the* truncated view $\mathcal{V}_G^{(T)}(u)$ *is the subgraph of $\mathcal{V}_G(u)$ induced by the nodes corresponding to paths of length up to and including T (the port label of the unique edge of each bottom-level node in $\mathcal{V}_G(u)$ is changed to 1).*

We refer the reader to Figure 4 for an illustration of the notions of view and truncated view. It is an easy property of $\mathcal{V}_G(u)$, as defined above, that every node of the view that corresponds to a path $u \rightsquigarrow v$ in G has degree equal to the degree of v in G. Moreover, the following can be proved by a straightforward induction on the length of the execution:

Proposition 1. *Let M be a mobile agent protocol and $\big((\mathsf{id}), G, (s), (x)\big)$ be an implicit input with a single agent, where G is not a tree. Then, for any integer $T \geq 0$, the sequences of configurations of the agent in the length-T prefixes of the two executions $M\big((\mathsf{id}), G, (s), (x)\big)$ and $M\big((\mathsf{id}), H, (r), (x)\big)$ are identical, where H is a graph obtained by attaching an arbitrary labeled graph to the bottom-level nodes of $\mathcal{V}_G^{(T+1)}(s)$, and r is the root of $\mathcal{V}_G^{(T+1)}(s)$. In particular, if $M\big((\mathsf{id}), G, (s), (x)\big)$ terminates after T steps, then the agent concludes its execution on H in T steps as well, with the same decision, and it only visits nodes that belong to the first T levels of the view $\mathcal{V}_G^{(T+1)}(s)$.*

2.2. Mobile agent decision problems

Definition 3 ([19]). *A* mobile agent decision problem *on anonymous graphs is a set Π of instances $(G, \vec{s}, \vec{x})$, where G is a graph, $\vec{s}$ is a non-empty list of nodes of G, and $\vec{x}$ is a list of strings with $|\vec{x}| = |\vec{s}|$, which satisfies the following closure property: For every instance $(G, \vec{s}, \vec{x})$ and for every automorphism α of G that preserves port numbers, $(G, \vec{s}, \vec{x}) \in \Pi$ if and only if $\big(G, \alpha(\vec{s}), \vec{x}\big) \in \Pi$.*[2]

[2]Note that this closure property is equivalent to the one used in [19], although the two are syntactically different due to notational differences.

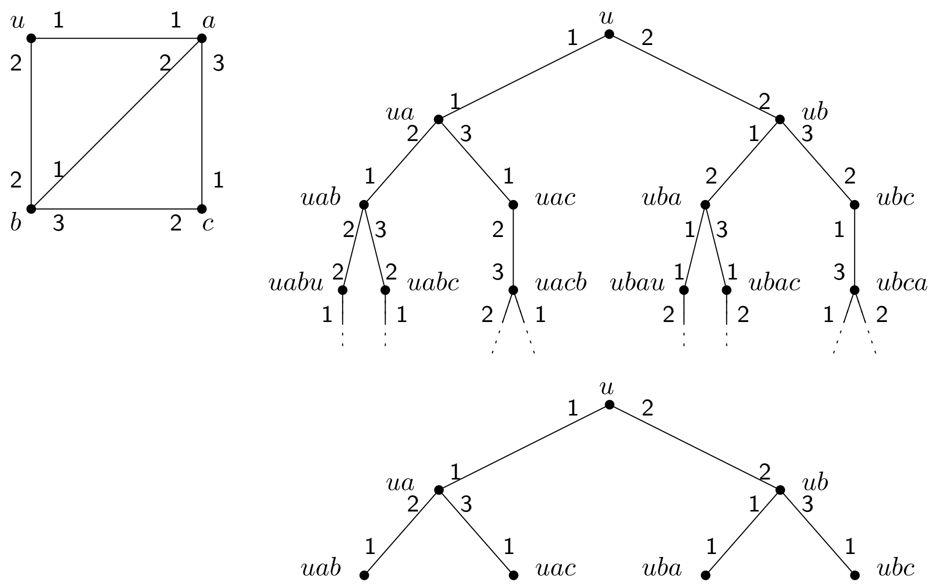


Figure 4: *Top left:* A port-labeled graph G. *Top right:* The first few levels of the view $\mathcal{V}_G(u)$. *Bottom:* The truncated view $\mathcal{V}_G^{(2)}(u)$.

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 $\overline{\Pi} = \{(G, \vec{s}, \vec{x}) : |\vec{s}| = |\vec{x}| \text{ and } (G, \vec{s}, \vec{x}) \notin \Pi\}$. It is easy to check that the class of mobile agent decision problems on anonymous graphs as defined in Definition 3 is closed under the usual set-theoretic operations:

Proposition 2. *If* Π, Π' *are mobile agent decision problems, then* $\overline{\Pi}$*,* $\Pi \cup \Pi'$*, and* $\Pi \cap \Pi'$ *are also mobile agent decision problems.*

Some examples of decision problems are shown in Table 2.

Definition 4 ([19]). *A decision problem* Π *is* mobile agent decidable *if there exists a protocol* M *such that for all instances* $(G, \vec{s}, \vec{x})$*: if* $(G, \vec{s}, \vec{x}) \in \Pi$ *then* $\forall \vec{\mathsf{id}}\ M(\vec{\mathsf{id}}, G, \vec{s}, \vec{x}) \mapsto \mathsf{yes}$*, whereas if* $(G, \vec{s}, \vec{x}) \notin \Pi$ *then* $\forall \vec{\mathsf{id}}\ M(\vec{\mathsf{id}}, G, \vec{s}, \vec{x}) \mapsto \widehat{\mathsf{no}}$*. The class of all decidable problems is denoted by* MAD*.*

Definition 5 ([19]). *A decision problem* Π *is* mobile agent verifiable *if there exists a protocol* M *such that for all instances* $(G, \vec{s}, \vec{x})$*: If* $(G, \vec{s}, \vec{x}) \in \Pi$ *then* $\exists \vec{y}\ \forall \vec{\mathsf{id}}\ M\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{y} \rangle\big) \mapsto \mathsf{yes}$*, whereas if* $(G, \vec{s}, \vec{x}) \notin \Pi$ *then* $\forall \vec{y}\ \forall \vec{\mathsf{id}}\ M\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{y} \rangle\big) \mapsto \widehat{\mathsf{no}}$*. 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 $\vec{y}$

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

accompanied	$=$	$\{(G,\vec{s},\vec{x}) : \lvert\vec{s}\rvert > 1\}$
allempty	$=$	$\big\{(G,\vec{s},\vec{x}) : \forall i\ x_i = \varepsilon\big\}$
allhalting$_j$	$=$	$\big\{(G,\vec{s},\vec{x}) : \forall i\ x_i \in \mathsf{HP}_j\big\}$
consensus	$=$	$\big\{(G,\vec{s},\vec{x}) : \forall i,j\ x_i = x_j\big\}$
degree	$=$	$\big\{(G,\vec{s},\vec{x}) : \forall i\ \exists v\ d_v = x_i\big\}$
mineven	$=$	$\big\{(G,\vec{s},\vec{x}) : \min_i x_i \text{ is even}\big\}$
path	$=$	$\big\{(G,\vec{s},\vec{x}) : G \text{ is a path}\big\}$
someempty	$=$	$\big\{(G,\vec{s},\vec{x}) : \exists i\ x_i = \varepsilon\big\}$
somenodes-ub	$=$	$\big\{(G,\vec{s},\vec{x}) : \exists i\ x_i \geq \lvert V(G)\rvert\big\}$
teamsize	$=$	$\big\{(G,\vec{s},\vec{x}) : \forall i\ x_i = \lvert\vec{s}\rvert\big\}$
treesize	$=$	$\big\{(G,\vec{s},\vec{x}) : \forall i\ G \text{ is a tree of size } x_i\big\}$

provided to the group of agents. If we need to distinguish between the two, we will refer to $\vec{y}$ as a *certificate vector*. Finally, if X is a class of mobile agent decision problems, then $\mathsf{co\text{-}X} = \{\Pi : \overline{\Pi} \in \mathsf{X}\}$.

On occasion, we will need to express the fact that, given full knowledge of an instance $I = (G,\vec{s},\vec{x})$, the problem of deciding whether $I \in \Pi$ for some mobile agent decision problem Π belongs to the computability class $\mathcal{C}$, where $\mathcal{C}$ is a standard computability class such as $\mathsf{\Delta}^0_1$ or $\mathsf{\Sigma}^0_1$. Despite the small inconsistency in the formalism, caused by the fact that $\mathcal{C}$ contains languages that are subsets of $\Sigma^\star$ and not mobile agent decision problems as per Definition 3, we will still write $\Pi \in \mathcal{C}$ to mean that $\big\{\langle I\rangle : I \in \Pi\big\} \in \mathcal{C}$, where $\langle\cdot\rangle$ is a fixed encoding function of mobile agent decision problem instances as strings over $\Sigma^\star$.

Remark 1. Note that in [19], only decidable (in the centralized 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 $\mathsf{\Delta}^0_1$. For the purposes of this work, we do not impose this constraint.

3. Mobile agent decidability classes

A problem Π is in $\mathsf{co\text{-}MAD}$ if and only if it can be decided by a mobile agent protocol in a sense which is dual to that of Definition 4: 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 6. *A decision problem Π is in $\mathsf{MAD_s}$ if and only if there exists a protocol M such that for all instances $(G,\vec{s},\vec{x})$: if $(G,\vec{s},\vec{x}) \in \Pi$ then $\forall\vec{\mathsf{id}}\ M(\vec{\mathsf{id}},G,\vec{s},\vec{x}) \mapsto \mathsf{yes}$, whereas if $(G,\vec{s},\vec{x}) \notin \Pi$ then $\forall\vec{\mathsf{id}}\ M(\vec{\mathsf{id}},G,\vec{s},\vec{x}) \mapsto \mathsf{no}$.*

By definition, $\mathsf{MAD_s}$ is a subset of both MAD and $\mathsf{co\text{-}MAD}$ 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 [19], 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 $\mathsf{treesize}$. The problem was already shown to be in MAD in [19]. For the stronger property that $\mathsf{treesize} \in \mathsf{MAD_s}$, we need a modification of the protocol given in [19].

Proposition 3. $\mathsf{treesize} \in \mathsf{MAD_s}$.

Proof. On input x, each agent performs a DFS traversal of the graph while drawing a map of it. The agent performs $2x - 2$ steps or stops earlier if it is back at its initial position with a complete map, that is without unvisited edges. When the agent stops, if its map is complete, is a tree, and has exactly x nodes, then it performs another full visit of the tree. Otherwise, it rejects. If it does not meet any agent that has rejected, then it accepts. Otherwise, it rejects. Clearly, all the agents accept if the instance belongs to $\mathsf{treesize}$, and all the agents reject if the input graph is not a tree. Lastly, if the input graph is a tree, say with n nodes, but the instance does not belong to $\mathsf{treesize}$, then there exists at least one agent with an input different from n. This agent rejects within the $2n - 2$ first steps. Therefore, no other agent can accept. This protocol proves that $\mathsf{treesize} \in \mathsf{MAD_s}$. $\square$

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

Proposition 4. $\mathsf{allempty} \in \mathsf{MAD} \setminus \mathsf{MAD_s}$ *and* $\overline{\mathsf{allempty}} \in \mathsf{co\text{-}MAD} \setminus \mathsf{MAD_s}$.

Proof. It suffices to show that $\mathsf{allempty} \in \mathsf{MAD} \setminus \mathsf{MAD_s}$, since this implies immediately that $\overline{\mathsf{allempty}} \in \mathsf{co\text{-}MAD} \setminus \mathsf{MAD_s}$.

The problem is in MAD in view of the following protocol: on input x, each agent accepts if and only if $x = \varepsilon$. Now, suppose that $\mathsf{allempty} \in \mathsf{MAD_s}$ and let M be the corresponding mobile agent protocol. An agent that executes M with explicit input $(\mathsf{id}, \varepsilon)$ and implicit input $\big((\mathsf{id}), G, (s), (\varepsilon)\big)$, where G is a ring with an arbitrary port labeling and s is an arbitrary node of the ring (i.e., the agent is executing M with an empty input, alone on a ring), must accept, say after T_1 steps. On the other hand, an agent that executes M with explicit input $(\mathsf{id}', 0)$, where $\mathsf{id}' \neq \mathsf{id}$, and implicit input $\big((\mathsf{id}'), G, (s), (0)\big)$ must reject, say after T_2 steps. Now, let G' be the graph obtained by connecting one of the bottom-level nodes of $\mathcal{V}_G^{(T_1+1)}(s)$ (cf. Definition 1) to one of the bottom-level nodes of $\mathcal{V}_G^{(T_2+1)}(s)$, via an edge with port number 2 at both endpoints. Let s_1, s_2 be the nodes of G' that respectively correspond to the roots of the two truncated view graphs that form G'. Consider the execution $M\big((\mathsf{id}, \mathsf{id}'), G', (s_1, s_2), (\varepsilon, 0)\big)$. Since not all inputs are empty and M is a $\mathsf{MAD_s}$ protocol, both agents should reject. However, by Proposition 1, one of them will accept, which is a contradiction. $\square$

As we mentioned, it holds by definition that $\mathsf{MAD_s}$ is included in both MAD and $\mathsf{co\text{-}MAD}$, i.e., $\mathsf{MAD_s} \subseteq \mathsf{MAD} \cap \mathsf{co\text{-}MAD}$. In fact, we can also prove the reverse inclusion and thus obtain $\mathsf{MAD_s} = \mathsf{MAD} \cap \mathsf{co\text{-}MAD}$: In order to obtain this result, we employ the meta-protocol of Section 4. We state it as a theorem without proof for the moment, and we refer the reader to Section 4.5 for a proof.

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

By Theorem 1, if $\mathsf{allempty}$ was included in $\mathsf{co\text{-}MAD}$, 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 $\mathsf{co\text{-}MAD}$. Symmetrically, $\overline{\mathsf{allempty}} \in \mathsf{co\text{-}MAD} \setminus \mathsf{MAD}$.

4. Interleaving multiple mobile agent protocols

In order to motivate the need to introduce a tool for interleaving the executions of multiple mobile agent protocols on the same instance, we start with an informal discussion of how to demonstrate the existence of a $\mathsf{MAD_s}$ protocol for a given problem Π, under the assumption that Π admits both a MAD protocol and a $\mathsf{co\text{-}MAD}$ protocol. Note that this would prove $\mathsf{MAD} \cap \mathsf{co\text{-}MAD} \subseteq \mathsf{MAD_s}$ and therefore it would establish Theorem 1. We will see that, to construct the $\mathsf{MAD_s}$ protocol, we need a way to execute the MAD and $\mathsf{co\text{-}MAD}$ protocols essentially in parallel on the same instance.

A first observation is that, in an execution of the MAD protocol, if an agent decides no, then that agent knows that the instance is a “no” instance and therefore it has the correct answer. On the other hand, if an agent decides yes, then it has no information on whether the instance is a “yes” or a “no” instance, because there is always the possibility that some other agent may decide no in the same execution. In this sense, we say that a no answer is *decisive* for an agent that executes the MAD protocol. Moreover, in an execution of the MAD protocol on a “no” instance at least one agent is guaranteed to give a decisive answer. Similarly, a yes answer is decisive for an agent that executes the $\mathsf{co\text{-}MAD}$ protocol, and in an execution of the $\mathsf{co\text{-}MAD}$ protocol on a “yes” instance at least one agent will give a decisive answer.

Now, let us assume, for the sake of argument, that the agents are able to execute both protocols in parallel on the same instance. In view of the above observation, at least one agent is guaranteed to give a decisive answer, either in the MAD protocol or in the $\mathsf{co\text{-}MAD}$ protocol. At that point, that agent knows whether the instance is a “yes” or “no” instance, and the only thing missing in order to construct the $\mathsf{MAD_s}$ protocol is a way for that agent to disseminate the correct answer to the other agents.

It is, therefore, necessary to have a tool that enables the execution of multiple mobile agent protocols on the same instance, and that also permits the mobile agents to make decisions based on the outcomes of these executions. In centralized computing, the well known *dovetailing* technique achieves this interleaving of different computations on the same input. Centralized 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, we develop in this section a generic mobile agent meta-protocol which enables the interleaving of a possibly infinite number of mobile agent protocols on the same instance. It, also, 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. This decision is taken independently by each agent, by locally executing a function, which is given as a parameter to the meta-protocol. We call this function a *local decider*.

Remark 2. In the example of proving $\mathsf{MAD} \cap \mathsf{co\text{-}MAD} \subseteq \mathsf{MAD_s}$, if an agent has nor reached a decisive answer so far in any of the two protocols, then the local decider instructs that agent to continue to the next phase. Otherwise, that agent terminates and acceps or rejects according to the decisive answer.

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 participate in 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*.

Remark 3. In the example of proving $\mathsf{MAD} \cap \mathsf{co\text{-}MAD} \subseteq \mathsf{MAD_s}$, the global decider, which is executed with full knowledge of the implicit input, simulates locally the MAD protocol and accepts if and only if all agents in the simulation accept.

In Section 4.1, we define formally the properties of global and local deciders and we present two auxiliary procedures that are used in the meta-protocol. Section 4.2 contains an informal description of the meta-protocol. Section 4.3 contains the complete pseudocode and Section 4.4 contains the proofs of some basic properties. Finally, in Section 4.5 we explain how we use the meta-protocol in proofs and we give two fairly simple applications (including the proof of Theorem 1).

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 whose executions will be interleaved. Let N_i, $i \geq 1$, denote the i-th protocol in such an enumeration. The functions f and g are the global and local deciders, respectively. These 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 $(\mathsf{id}, x; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$, 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 the current phase number T, an explicit input (id, x), and a list $(H_1, \dots, H_\sigma)$ of arbitrary length σ, where each H_j is the history of the partial execution of $N_j(\mathsf{id}, x; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ for at most T steps and $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ is an implicit input common for all histories $H_1, \dots, H_\sigma$. 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 $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ and for every T_0, there exists a $T \geq T_0$ and some i such that the local computation $g(T, \mathsf{id}_i, x_i, H_1, \dots, H_{\min(T, |\mathcal{N}|)})$ returns either **accept** or **reject**, where each H_j is an encoding of the execution of $N_j(\mathsf{id}_i, x_i; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ for at most 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 (and, therefore, that the maximum degree of the graph is at most R). More precisely, the agent first creates a map consisting of 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 $\textsc{Rdv}(R, \mathsf{id}_i)$ 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. Note that R is also an upper bound on the maximum degree, the diameter, and therefore also on the maximum radius of a ball in the graph. The Rdv procedure uses as a subroutine the following Explore-Ball procedure: An agent executing $\textsc{Explore-Ball}(R)$ 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 $\textsc{Explore-Ball}(R)$ at the same time from different nodes are synchronized and back at their starting positions after $B(R)$ steps.

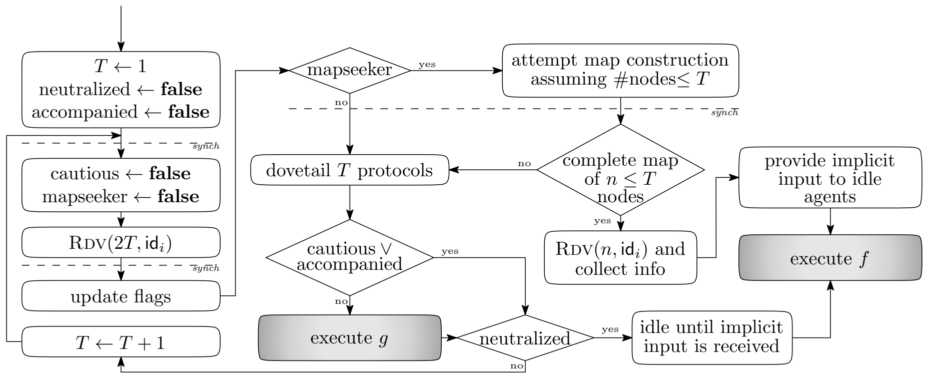
Now, for each bit of id_i, the Rdv procedure executes the following: If the bit is 0, the agent waits at s_i for $B(R)$ steps and then executes $\textsc{Explore-Ball}(R)$, whereas if the bit is 1, the agent first executes $\textsc{Explore-Ball}(R)$ and then waits on its starting position for $B(R)$ steps. After it exhausts the bits of id_i, the agent executes twice $\textsc{Explore-Ball}(R)$. 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

We start with an informal description of the meta-protocol and we give the complete pseudocode in Section 4.3.

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 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 (i.e., executing line 39). 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. 5):

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 (line 5). 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



$T \leftarrow 1$
neutralized $\leftarrow$ **false**
accompanied $\leftarrow$ **false**
synch
cautious $\leftarrow$ **false**
mapseeker $\leftarrow$ **false**
Rdv($2T$, id$_i$)
synch
update flags
$T \leftarrow T+1$
mapseeker
yes
no
attempt map construction assuming #nodes$\leq T$
synch
dovetail T protocols
no
complete map of $n \leq T$ nodes
yes
cautious $\vee$ accompanied
yes
no
Rdv(n, id$_i$) and collect info
provide implicit input to idle agents
execute f
execute g
neutralized
yes
no
idle until implicit input is received

Figure 5: High-level flowchart of the meta-protocol of Section 4.

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 (line 17).

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 (in line 5). Then, it attempts to create a map of the graph by executing Create-Map(T) and returns to s_i. Overall, this takes at most $4T^4 + 4T$ steps. Meanwhile, non-mapseeker agents wait for $4T^4 + 4T$ steps. All agents synchronize at this point (line 24).

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 Rdv(n, id$_i$), which guarantees that, finally, it is located at s_i and has met all other agents executing Rdv, as well as all halted and idle agents. While executing Rdv, the agent collects starting position and input information from all other mapseeker, halted, and idle agents that it encounters. At this point, each mapseeker does a final exploration of the graph to provide its knowledge to the idle agents. Then, after concluding this last exploration, each mapseeker executes f (line 28) 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, line 33). 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 $g(T, \mathsf{id}, x, H_1, \ldots, H_{\min(T,|\mathcal{N}|)})$, where H_j is the history of the T-step execution of N_j with explicit input (id, x). 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 $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$, even though some of the agents may have halted or become idle in earlier phases (Lemma 3 and Corollary 1) (line 37).

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 (line 40).

4.3. Pseudocode

Explicit input: (id, x). *Implicit input:* $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$.

```
 1: $T \leftarrow 1$
 2: neutralized, accompanied ← **false**
 3: **loop**
        ▶ *all participating agents are synchronized at this point*
 4:     cautious, mapseeker ← **false**

 5:     Execute RDV($2T$, id) for $2(T+1)B(2T)$ steps
 6:     **if** met an agent **then**
 7:         accompanied ← **true**
 8:     **end if**
 9:     **if** met a participating agent with lexicographically larger ID **then**
10:         neutralized ← **true**
11:     **end if**
12:     **if** found a halted or idle agent **then**
13:         mapseeker ← **true**
14:     **else if** found a node with degree greater than $2T$ or $|\mathsf{id}| > T$ **then**
15:         cautious ← **true**
16:     **end if**
17:     Wait until $2(T+1)B(2T)$ steps have passed since last synchronization
        point
        ▶ *all participating agents are synchronized at this point*

18:     **if** mapseeker **then**
19:         Move to the halted or idle agent with the lexicographically smallest
            ID among those found during step 5
20:         Execute CREATE-MAP($T$)
21:         Return to starting node and wait until $4T^4 + 4T$ steps have passed
            since last synchronization point
22:     **else**
23:         Wait for $4T^4 + 4T$ steps
```

```
24:     end if
        ▶ all participating agents are synchronized at this point

25:     if mapseeker and full map of $n$ nodes is known and $T \geq n$ then
26:         Execute Rdv($n$, id), while collecting starting position and input information from mapseeker, halted, and idle agents
27:         Provide the implicit input to idle agents and return to starting node

28:         Execute $f(\mathsf{id}, x; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$          ▷ Local computation
29:     end if

30:     for $j \leftarrow 1$ to $\min\big(T, |\mathcal{N}|\big)$ do
31:         Execute $T$ steps of $N_j$ with explicit input $(\mathsf{id}, x)$, without halting
32:         Let $H_j$ be the corresponding history
33:         Return to starting node
            ▶ all participating agents are synchronized at this point
34:     end for
35:     if cautious = accompanied = false then
36:         Execute $g(T, \mathsf{id}, x, H_1, \ldots, H_{\min(T,|\mathcal{N}|)})$          ▷ Local computation
37:     end if

38:     if neutralized then
39:         Remain idle until the implicit input $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ is provided by a mapseeker
40:         Execute $f(\mathsf{id}, x; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$          ▷ Local computation
41:     end if
42:     $T \leftarrow T + 1$
43: end loop
```

4.4. Properties of the meta-protocol

Lemma 1. *In each phase, either all or none of the participating agents (i.e., non-halted and non-idle) execute f (line 28).*

Proof. We show that if the condition of line 25 is true for one participating agent, then it is true for all participating agents. Indeed, since there exists at least one mapseeker agent, there must exist at least one halted or idle agent. Moreover, since the size of G is n and $T \geq n$, every participating agent will locate at least one halted or idle agent while executing Rdv($2T$) (line 5), and thus all participating agents will become mapseeker agents. Consequently, since $T \geq n$, all of them will obtain the full map of the graph while executing Create-Map(T) (line 20) and also all of them will obtain the starting position and input information from all halted and idle agents. □

Lemma 2. *Any agent that executes f (line 28 or 40) has full knowledge of the implicit input $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$.*

Proof. By the properties of the Rdv procedure, since $T \geq n$, all mapseeker agents will meet each other while executing line 26, and thus they will proceed to execute f at line 28 with full knowledge of the map of the graph and the starting positions and inputs of all agents. Moreover, an agent that executes f at line 40 has received the implicit input from a mapseeker who is executing line 27, therefore it knows the full implicit input. □

Lemma 3. *If an agent i executes g (line 36) during phase T, then no other agent's starting node is at distance $2T$ or less from s_i.*

Proof. Since cautious = **false**, the execution of Rdv$(2T, \mathsf{id})$ for $2(T+1)B(2T)$ steps at line 5 did not reveal any node of degree greater than $2T$ and, moreover, $|\mathsf{id}| \leq T$. By the properties of Rdv, this implies that the agent completely explored the $2T$-ball around s_i at least once while any other participating agent was sitting on its respective starting node. Therefore, since, in addition, we have accompanied = **false**, the lemma follows. □

By Lemma 3, we obtain the following corollary:

Corollary 1. *Any agent i that executes g (line 36) has histories which correspond to the correct histories of $N_j(\mathsf{id}_i, x_i; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ 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 $\mathcal{P}_{\mathcal{N},f,g}(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ by executing either f or g. Moreover, each agent i halts on its starting node s_i.*

Proof. Let n be the size of G. We first show that at least one agent will halt or become idle. By Corollary 1 and by the local decider property of g, there exists a phase $T \geq \max(n, \max_i |\mathsf{id}_i|)$ in which some agent i would halt if it executed g at line 36. By the choice of T, this agent cannot have the cautious flag set. Therefore, it either executes g, and therefore it halts, or its accompanied flag is set. But then, it either met a participating agent while executing the Rdv at line 5 resulting in one of the two agents becoming neutralized and thus idle at the end of the phase, or it met an agent which had already halted or become idle in an earlier phase.

Now, let T_0 be the phase in which the first agent halts or becomes idle and let $T_1 = \max(T_0+1, n, \max_i |\mathsf{id}_i|)$. During phase T_1, all participating agents will explore the whole graph while executing Explore-Ball at line 5, and therefore they will all locate the agent that halted or became idle during phase T_0 and they will set their mapseeker flag. Subsequently, they will all obtain the full map of G while executing Create-Map at line 20, and finally they will all execute f at line 28 after providing the correct implicit input to all idle agents, which will execute f at line 40. By the global decider property of f, all agents will then halt.

It remains to show that each agent i halts on s_i. By inspection of the meta-protocol, an agent halts only by executing f or g, and these functions are only executed when the agent is on s_i. $\square$

4.5. Applications 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 typically 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.

We will employ the meta-protocol in subsequent sections to prove Theorems 3, 4, 5, and 7. In order to demonstrate the basic layout of such proofs, we now give the proof of Theorem 1 from Section 3.

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

Proof. It suffices to show that $\mathsf{MAD} \cap \mathsf{co\text{-}MAD} \subseteq \mathsf{MAD_s}$. Let $\Pi \in \mathsf{MAD} \cap \mathsf{co\text{-}MAD}$ and let N_1 be a MAD protocol for Π and N_2 be a $\mathsf{co\text{-}MAD}$ protocol for Π. We use the meta-protocol $\mathcal{P}_{\mathcal{N},f,g}$ with $\mathcal{N} = (N_1, N_2)$ and deciders f and g defined as follows:

- To compute $f(\mathsf{id}, x; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$: Simulate locally the execution of N_1 on the implicit input $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$. If at least one agent rejects, return **reject**. Otherwise, return **accept**.
- To compute $g(T, \mathsf{id}, x, H_1, H_2)$: If H_1 is the history of an execution of N_1 which terminates in a rejecting state, then return **reject**. If H_2 is the history of an execution of N_2 which terminates in an accepting state, then return **accept**. Otherwise, return **continue**.

The function f is clearly a global decider. To see that g is a local decider, note that any given instance is either a "yes" instance, in which case at least one agent executing N_2 will accept, or it is a "no" instance, in which case at least one agent executing N_1 will reject. In any case, given sufficiently long histories H_1, H_2, the execution of g will result in a return value in $\{\textbf{accept}, \textbf{reject}\}$ for at least one agent.

It remains to show that all agents terminate in the correct state. By Lemma 4, each agent halts by executing either f or g. By Lemma 2, an agent that halts by executing f will execute it with the correct full knowledge of the implicit input, and therefore, by definition of MAD, the local simulation of N_1 will result in at least one agent rejecting if and only if the instance is a "no" instance. Therefore, the agent executing f will reject if and only if $(G, \vec{s}, \vec{x}) \notin \Pi$.

Finally, let i be an agent that halts by executing g. By Corollary 1, when it halts, H_1 and H_2 are the correct histories of $N_1(\mathsf{id}_i, x_i; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ and $N_2(\mathsf{id}_i, x_i; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$, respectively. If the agent accepts, then by definition of g, H_2 terminates in an accepting state, therefore by definition of $\mathsf{co\text{-}MAD}$, $(G, \vec{s}, \vec{x}) \in \Pi$. On the other hand, if the agent rejects, then by definition of g, H_1 terminates in a rejecting state, therefore by definition of MAD, $(G, \vec{s}, \vec{x}) \notin \Pi$. We conclude that all agents give the correct answer and thus $\Pi \in \mathsf{MAD_s}$. □

We conclude this section with a second fairly simple application of the meta-protocol, which may be of independent interest: We show that, without loss of generality, we can assume that any protocol terminates by having all agents back on their respective starting nodes.

Theorem 2. *Let M be a terminating mobile agent protocol. Then, there exists a mobile agent protocol $M^{(r)}$ such that for every implicit input $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$, each agent i halts in the same state (accepting or rejecting) under $M^{(r)}(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ as under $M(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ and, furthermore, it halts on its starting node s_i.*

Proof. Let $\mathcal{N} = (M)$. Let $f(\mathsf{id}, x; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ be defined as follows: Simulate $M(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ locally and halt in the same state as the agent with ID id. Let $g(T, \mathsf{id}, x, H)$ be defined as follows: Examine the history H and, if it terminates, then accept or reject according to the result of H. Otherwise, continue.

Since we assume that in $M(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ all agents halt, it is straightforward to verify that f is a global decider and g is a local decider. Now, let us examine what happens when $\mathcal{P}_{\mathcal{N},f,g}$ is executed:

By Lemma 2, if an agent halts by executing f, then it indeed terminates in the same state as if it had executed M. By the local decider property of g and Corollary 1, if an agent halts by executing g, then again it halts in the same state as if it had executed M. Finally, by Lemma 4, all agents halt and, in fact, each halts on its respective starting node. □

5. Mobile agent verifiability classes

5.1. Verification with unanimity

Similarly to $\mathsf{MAD_s}$, we will consider an analogous "strict" version of MAV:

Definition 7. *A decision problem* Π *is in* $\mathsf{MAV_s}$ *if and only if there exists a protocol* M *such that for all instances* $(G, \vec{s}, \vec{x})$*: if* $(G, \vec{s}, \vec{x}) \in \Pi$ *then* $\exists \vec{y}\, \forall \vec{\mathsf{id}}\ M\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{y} \rangle\big) \mapsto \mathsf{yes}$*, whereas if* $(G, \vec{s}, \vec{x}) \notin \Pi$ *then* $\forall \vec{y}\ \forall \vec{\mathsf{id}}\ M\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{y} \rangle\big) \mapsto \mathsf{no}$.

Note that, for "yes" instances, there can be certificates that lead some agents to accept and some agents to reject. The decision is thus not necessarily unanimous for all certificates in all instances.

By definition, $\mathsf{MAV_s} \subseteq \mathsf{MAV}$. Moreover, $\mathsf{MAV} \subseteq \mathbf{\Sigma}^0_1$, 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 $\mathsf{co\text{-}MAV_s} \subseteq \mathsf{co\text{-}MAV} \subseteq \mathbf{\Pi}^0_1$.

There exist several nontrivial problems in $\mathsf{MAV_s}$ and $\mathsf{co\text{-}MAV_s}$ (Proposition 5). Furthermore, we can show that MAV is a strict superset of $\mathsf{MAV_s}$ and, as a corollary, $\mathsf{co\text{-}MAV}$ is a strict superset of $\mathsf{co\text{-}MAV_s}$ (Proposition 6).

Proposition 5. $\mathsf{accompanied} \in \mathsf{MAV_s}$ *and* $\mathsf{consensus} \in \mathsf{co\text{-}MAV_s}$.

Proof. For each problem we give only the corresponding verification protocol and we omit the straightforward correctness proof.

A $\mathsf{MAV_s}$ protocol for $\mathsf{accompanied}$ is the following: On explicit input $\big(\mathsf{id}, \langle x, y \rangle\big)$, each agent interprets y as an integer representing the size of the graph. Then, it executes $\textsc{Rdv}(y, \mathsf{id})$. After $\textsc{Rdv}$ concludes, the agent accepts if it has met any other agent, otherwise it rejects.

A $\mathsf{co\text{-}MAV_s}$ protocol for $\mathsf{consensus}$ is the following: On explicit input $\big(\mathsf{id}, \langle x, y \rangle\big)$, each agent interprets y as an integer representing the size of the graph. Then, it executes $\textsc{Rdv}(y, \mathsf{id})$, while storing all inputs from the agents it encounters, including its own. After $\textsc{Rdv}$ concludes, the agent accepts if all the inputs it has recorded are the same, otherwise it rejects. □

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

Proof. The problem is in MAV in view of the following protocol: The certificate y provided to each agent is interpreted as a sequence of port numbers to follow. If, after following y, the agent reaches a node of degree equal to its input, then it accepts, otherwise it rejects.

Now, suppose that $\mathsf{degree} \in \mathsf{MAV_s}$ and let M be the corresponding protocol. An agent that executes M with explicit input $\big(\mathsf{id}, \langle 3, \varepsilon \rangle\big)$ and implicit input $\big((\mathsf{id}), G, (s), (\langle 3, \varepsilon \rangle)\big)$, where G is an arbitrarily labeled ring and s is an arbitrary node of G, must reject, say after T_1 steps, because the ring does not contain a node of degree 3. On the other hand an agent that executes M with explicit input $\big(\mathsf{id}', \langle 2, y \rangle\big)$, where $\mathsf{id}' \neq \mathsf{id}$, and implicit input $\big((\mathsf{id}'), G, (s), (\langle 2, y \rangle)\big)$ must clearly accept for some certificate $y = y^\star$, say after T_2 steps. Let G' be the graph resulting from connecting one bottom-level node of $\mathcal{V}_G^{(T_1+1)}(s)$ (cf. Definition 1) to one bottom-level node of $\mathcal{V}_G^{(T_2+1)}(s)$ (the new edge receives port number 2 at both endpoints). By Proposition 1, in the execution

$M\big((\mathsf{id},\mathsf{id}'),G',(s_1,s_2),(\langle 3,\varepsilon\rangle,\langle 2,y^\star\rangle)\big)$, where s_1 and s_2 are the respective roots of the truncated views that form G', one of the agents will accept, whereas both should reject because G' does not contain any node of degree 3 and hence this is a "no" instance of the problem.[3] This contradicts the existence of a $\mathsf{MAV_s}$ protocol for degree.

Furthermore, suppose that $\mathsf{degree} \in \mathsf{co\text{-}MAV}$ and let M be the corresponding protocol. An agent that executes M with explicit input $\big(\mathsf{id},\langle 1,y_1\rangle\big)$ and implicit input $\big((\mathsf{id}),G,(s),(\langle 1,y_1\rangle)\big)$, where G is an arbitrarily labeled ring and s is an arbitrary node of G, must reject for some certificate $y_1 = y_1^\star$, say after T_1 steps, because the ring does not contain a node of degree 1. Similarly, there exists $y_2^\star$ such that an agent that executes M with explicit input $\big(\mathsf{id}',\langle 1,y_2^\star\rangle\big)$, where $\mathsf{id}' \neq \mathsf{id}$, and implicit input $\big((\mathsf{id}'),G,(s),(\langle 1,y_2^\star\rangle)\big)$ rejects, say after T_2 steps. Let G' be the graph resulting from connecting one bottom-level node of $\mathcal{V}_G^{(T_1+1)}(s)$ (cf. Definition 1) to one bottom-level node of $\mathcal{V}_G^{(T_2+1)}(s)$ (the new edge receives port number 2 at both endpoints). By Proposition 1, in the execution $M\big((\mathsf{id},\mathsf{id}'),G',(s_1,s_2),(\langle 1,y_1^\star\rangle,\langle 1,y_2^\star\rangle)\big)$, where s_1 and s_2 are the respective roots of the truncated views that form G', both agents will reject. However, this is a "yes" instance of the problem, because it contains two nodes of degree 1 (the remaining bottom nodes of the two truncated views[4]). This contradicts the existence of a $\mathsf{co\text{-}MAV}$ protocol for degree. $\square$

Proposition 6 also separates MAV from $\mathsf{co\text{-}MAV}$. In order to separate $\mathsf{\Sigma}_1^0$ from MAV and $\mathsf{\Pi}_1^0$ from $\mathsf{co\text{-}MAV}$, we observe that the $\mathsf{teamsize}$ problem, which is clearly in $\mathsf{\Delta}_1^0 = \mathsf{\Sigma}_1^0 \cap \mathsf{\Pi}_1^0$, is neither in MAV nor in $\mathsf{co\text{-}MAV}$.

Proposition 7. $\mathsf{teamsize} \in \mathsf{\Delta}_1^0 \setminus (\mathsf{MAV} \cup \mathsf{co\text{-}MAV})$.

Proof. Suppose, for the sake of contradiction, that $\mathsf{teamsize} \in \mathsf{MAV}$ and let M be the corresponding protocol. An agent that executes M with explicit input $\big(\mathsf{id},\langle 1,y\rangle\big)$ and implicit input $\big((\mathsf{id}),G,(s),(\langle 1,y\rangle)\big)$, where G is an arbitrarily labeled ring and s is an arbitrary node of G, must accept for some $y = y^\star$, say after T_1 steps. Similarly, an agent with explicit input $\big(\mathsf{id}',\langle 1,y^\star\rangle\big)$, where $\mathsf{id}' \neq \mathsf{id}$, and implicit input $\big((\mathsf{id}'),G,(s),(\langle 1,y^\star\rangle)\big)$ must accept, say after T_2 steps. Let G' be the graph obtained by connecting one bottom-level node of $\mathcal{V}_G^{(T_1+1)}(s)$ (cf. Definition 1) to one bottom-level node of $\mathcal{V}_G^{(T_2+1)}(s)$ (the new edge receives port number 2 at both endpoints). By Proposition 1, if s_1 and s_2 are the roots of the truncated views forming G', in the execution $M\big((\mathsf{id},\mathsf{id}'),G',(s_1,s_2),(\langle 1,y^\star\rangle,\langle 1,y^\star\rangle)\big)$ both agents will accept, whereas at least one should reject. This contradicts the existence of a MAV protocol for $\mathsf{teamsize}$.

[3]Recall that the degree of each node in the view is equal to the degree of some node in the original graph, apart from the leaves of the view which of course have degree 1.

[4]Note that $\mathcal{V}_G^{(T_1+1)}(s)$ and $\mathcal{V}_G^{(T_2+1)}(s)$ are simply paths of length $2(T_1+1)$ and $2(T_2+1)$, respectively, and G' is a path of length $2(T_1+1)+2(T_2+1)+1$.

By a completely symmetric argument, with input 2 instead of 1, we also obtain a contradiction under the assumption that $\mathsf{teamsize} \in \mathsf{co\text{-}MAV}$. $\square$

5.2. Decision problems with "no" certificates

In classical computability, the class $\mathsf{\Pi}_1^0 = \mathsf{co\text{-}\Sigma}_1^0$ 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 $\mathsf{\Sigma}_1^0$, $\mathsf{co\text{-}MAV}$ is not quite the analogue of $\mathsf{\Pi}_1^0$. Problems in $\mathsf{co\text{-}MAV}$ 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 $\mathsf{co\text{-}MAV}'$, 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 8. *A decision problem* Π *is in* MAV' *if and only if there exists a protocol* M *such that for all instances* $(G, \vec{s}, \vec{x})$*: if* $(G, \vec{s}, \vec{x}) \in \Pi$ *then* $\exists \vec{y}\ \forall \vec{\mathsf{id}}\ M\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{y} \rangle\big) \mapsto \widehat{\mathsf{yes}}$*, whereas if* $(G, \vec{s}, \vec{x}) \notin \Pi$ *then* $\forall \vec{y}\ \forall \vec{\mathsf{id}}\ M\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{y} \rangle\big) \mapsto \mathsf{no}$.

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

Proof. It suffices to show that $\mathsf{MAV}' \subseteq \mathsf{MAV_s}$. Let $\Pi \in \mathsf{MAV}'$ and let M be a MAV' protocol for Π. We will use the meta-protocol $\mathcal{P}_{\mathcal{N},f,g}$ of Section 4 in order to give a $\mathsf{MAV_s}$ protocol for Π. The explicit input to the meta-protocol will be of the form $\big(\mathsf{id}, \langle x, \langle y_1, y_2 \rangle \rangle\big)$, where x is the input string and $\langle y_1, y_2 \rangle$ is the certificate. The first part of the certificate, y_1, will be interpreted as the number of nodes in the graph. The second part of the certificate, y_2, will be interpreted as the certificate for the MAV' protocol. The idea is that all agents perform the MAV' protocol with certificate y_2 and, if that protocol instructs an agent to accept (which is a *decisive* answer for the MAV' protocol), then the agent accepts. On the other hand, if the MAV' protocol instructs an agent to reject, then it rejects only after y_1 meta-protocol phases have been executed. This ensures that, in a "yes" instance, an agent that would reject in view of the MAV' protocol will have the chance to meet some other agent that will have accepted.

Let $\mathcal{N} = (N)$, where N is the following protocol: Protocol N executes M with explicit input $\big(\mathsf{id}, \langle x, y_2 \rangle\big)$. For the deciders of the meta-protocol, we compute $f\big(\mathsf{id}, \langle x, \langle y_1, y_2\rangle\rangle ; \vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \langle \vec{y_1}, \vec{y_2}\rangle\rangle\big)$ as follows: simulate locally M on implicit input $\big(\vec{\mathsf{id}}, G, \vec{s}, \langle x, \vec{y_2}\rangle\big)$. If some agent in the simulation accepts, then accept. Otherwise, reject. Finally, we compute $g\big(T, \mathsf{id}, \langle x, \langle y_1, y_2\rangle\rangle, H\big)$ as follows: If H is a history that ends by accepting, then accept. If it ends by rejecting and T (the current phase number of the meta-protocol) is larger than or equal to y_1, then reject. In any other case, continue (g is not decisive yet).

The global decider property clearly holds for f, as it always either accepts or rejects. For g, the local decider property holds because eventually T will become so large that all agents terminate M on one hand, and T will be larger than all values y_1 on the other hand. At that point g will return a decisive result (either accept or reject).

It remains to show that $\mathcal{P}_{\mathcal{N},f,g}$ is indeed a $\mathsf{MAV_s}$ protocol for Π. Let $(G, \vec{s}, \vec{x}) \in \Pi$. By definition of MAV', there exists a certificate $\vec{y}^\star$ such that for all $\vec{\mathsf{id}}$, at least one agent accepts under the execution $M\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{y}^\star\rangle\big)$. Let $\vec{n}$ be a vector with $|\vec{n}| = |\vec{y}^\star|$ and $n_i = |V(G)|$ for all i. We will show that, for any ID vector $\vec{\mathsf{id}}$, the execution $\mathcal{P}_{\mathcal{N},f,g}\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \langle \vec{n}, \vec{y}^\star\rangle\rangle\big)$ results in all agents accepting. Consider an agent that halts by executing f. By Lemma 2, the agent knows the full implicit input, and by the property of $\vec{y}^\star$, at least one agent will accept in the simulation. Therefore, the agent that halts by executing f will accept. Now, consider an agent that halts by executing g. By definition of g, and from the fact that g is only executed when the cautious and accompanied flags are not set, that agent may reject only if it is alone in the graph and, by Corollary 1, only if it rejects under the execution $M\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{y}^\star\rangle\big)$, which is a contradiction. Therefore, any agent that halts by executing g will accept. By Lemma 4 all agents halt, and thus, for "yes" instances, all agents accept under $\mathcal{P}_{\mathcal{N},f,g}$.

Finally, let $(G, \vec{s}, \vec{x}) \notin \Pi$. By definition of MAV', for all certificates $\vec{y}$ and all $\vec{\mathsf{id}}$, all agents reject under the execution $M\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{y}\rangle\big)$. Therefore, by Lemma 2, any agent that halts by executing f will reject. Moreover, by Corollary 1, any agent that halts by executing g must also reject. Finally, by Lemma 4, all agents reject. $\square$

In view of Theorem 3, it follows that $\mathsf{MAD} \subseteq \mathsf{MAV} \cap \mathsf{co\text{-}MAV}$ and $\mathsf{co\text{-}MAD} \subseteq \mathsf{MAV} \cap \mathsf{co\text{-}MAV}$. We separate $\mathsf{MAV} \cap \mathsf{co\text{-}MAV}$ from MAD and $\mathsf{co\text{-}MAD}$ with the problem $\mathsf{mineven}$. In fact, we prove a stronger separation between $\mathsf{MAV} \cap \mathsf{co\text{-}MAV}$ and $\mathsf{MAV_s} \cup \mathsf{co\text{-}MAV_s} \supseteq \mathsf{MAD} \cup \mathsf{co\text{-}MAD}$:

Proposition 8. $\mathsf{mineven} \in (\mathsf{MAV} \cap \mathsf{co\text{-}MAV}) \setminus (\mathsf{MAV_s} \cup \mathsf{co\text{-}MAV_s})$.

Proof. We prove $\mathsf{mineven} \in \mathsf{MAV} \setminus \mathsf{MAV_s}$. A MAV protocol for $\mathsf{mineven}$ is the following: The certificate y_i provided to each agent is interpreted as the number of nodes of the graph. Then, the agents execute a protocol which guarantees that, if they have received the correct certificate, they will all meet each other at least once. For instance, this is guaranteed if each agent executes $\textsc{Rdv}(y_i, \mathsf{id}_i)$, where $\textsc{Rdv}$ is one of the auxiliary procedures for the meta-protocol (see Section 4.1). At the same time, the agent collects the input values of other agents that it meets. Finally, it accepts if and only if the minimum of all collected input values (including its own) is even. For "yes" instances, there exists a certificate for which each agent meets all other agents, collects their inputs, and verifies correctly that the minimum input is even. On the other hand, for "no" instances, for every certificate, each agent collects a certain subset of the input values of the agents. The agent that received the minimum input (which is odd, since we are in a "no" instance), will always have a subset in which the minimum value is odd. Therefore, that agent will reject.

Furthermore, suppose that $\mathsf{mineven} \in \mathsf{MAV_s}$ and let M be the corresponding protocol. An agent that executes M with explicit input $\big(\mathsf{id}, \langle 1, y_1\rangle\big)$ and implicit input $\big((\mathsf{id}), G, (s), (\langle 1, y_1\rangle)\big)$, where G is an arbitrarily labeled ring and s is an arbitrary node of G, must reject for every y_1, since this is a "no" instance. We fix a certificate y_1 and let T_1 be the number of steps until the agent rejects. On the other hand, an agent that executes M with explicit input $\big(\mathsf{id}', \langle 2, y_2\rangle\big)$ and implicit input $\big((\mathsf{id}'), G, (s), (\langle 2, y_2\rangle)\big)$ must accept for some certificate $y_2 = y_2^\star$, say after T_2 steps. Let G' be the graph obtained by connecting one bottom-level node of $\mathcal{V}_G^{(T_1+1)}(s)$ (cf. Definition 1) to one bottom-level node of $\mathcal{V}_G^{(T_2+1)}(s)$ (the new edge receives port number 2 at both endpoints). By Proposition 1, if s_1 and s_2 are the roots of the truncated views forming G', in the execution $M\big((\mathsf{id}, \mathsf{id}'), G', (s_1, s_2), (\langle 1, y_1\rangle, \langle 2, y_2^\star\rangle)\big)$ one agent will accept, whereas both should reject. This contradicts the existence of a $\mathsf{MAV_s}$ protocol for $\mathsf{mineven}$.

By similar arguments, we can prove that $\mathsf{minodd} \in \mathsf{MAV} \setminus \mathsf{MAV_s}$, where $\mathsf{minodd} = \big\{(G, \vec{s}, \vec{x}) : \min_i x_i \text{ is odd}\big\}$. Therefore, since $\mathsf{minodd} = \overline{\mathsf{mineven}}$, we obtain $\mathsf{mineven} \in \mathsf{co\text{-}MAV} \setminus \mathsf{co\text{-}MAV_s}$. □

5.3. 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 3, $\mathsf{MAD} \subseteq \mathsf{co\text{-}MAV_s}$. 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 4 that, in fact, $\mathsf{MAD_s} = \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV_s}$. Furthermore, from the definitions and Theorem 3, 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 5 below. The proof of Theorem 4 (resp. Theorem 5) 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 4. $\mathsf{MAD_s} = \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV_s}$.

Proof. It suffices to show that $\mathsf{MAV_s} \cap \mathsf{co\text{-}MAV_s} \subseteq \mathsf{MAD_s}$. Let N_1 be a $\mathsf{MAV_s}$ protocol for Π and let N_2 be a $\mathsf{co\text{-}MAV_s}$ protocol for Π, where $\Pi \in \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV_s}$. In order to show that $\Pi \in \mathsf{MAD_s}$, we will use the meta-protocol $\mathcal{P}_{\mathcal{N},f,g}$ with a collection $\mathcal{N}$ defined as follows:

For $\rho \geq 1$, let $\mathcal{T}_\rho$ denote the set of all functions that map strings of length up to ρ to strings of length up to ρ and let $(T_{\rho,j})_{1\leq j\leq|\mathcal{T}_\rho|}$ be a fixed enumeration of $\mathcal{T}_\rho$. Now, for $i \in \{1, 2\}$, we define the protocol $N_i^{\rho,j}$:

- On explicit input (id, x), $N_i^{\rho,j}$ first checks if $|\mathsf{id}| \leq \rho$. If so, then it computes $y = T_{\rho,j}(\mathsf{id})$, otherwise it sets $y = \varepsilon$. Finally, it executes N_i with explicit input $\big(\mathsf{id}, \langle x, y\rangle\big)$.

Let $\mathcal{N} = \big\{N_i^{\rho,j} : i \in \{1, 2\}, \rho \geq 1, 1 \leq j \leq |\mathcal{T}_\rho|\big\}$. Note that $\mathcal{N}$ is an infinite set of mobile agent protocols, but it admits an effective enumeration.

Remark 4. The following holds by construction: For every $k \geq 1$, every list of distinct strings $\vec{\mathsf{id}}$, every list of strings $\vec{y}$ with $|\vec{\mathsf{id}}| = |\vec{y}| = k$, and every $\rho \geq \rho_0$, where ρ_0 is the maximum of all lengths of strings in $\vec{\mathsf{id}}$ and $\vec{y}$, there exists j in the range $1 \leq j \leq |\mathcal{T}_\rho|$ such that $T_{\rho,j}(\vec{\mathsf{id}}) = \vec{y}$.

We now define the functions f and g of the meta-protocol.

- To compute $f(\mathsf{id}, x; \vec{\mathsf{id}}, G, \vec{s}, \vec{x})$: Simulate locally, by classical dovetailing, all protocols in $\mathcal{N}$ on the implicit input $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$. If, for some pair (ρ, j), $N_1^{\rho,j}$ results in all agents accepting, then accept. If, for some pair (ρ, j), $N_2^{\rho,j}$ results in all agents rejecting, then reject.
- To compute $g(T, \mathsf{id}, x, H_1, \ldots, H_\sigma)$: If, for some (i, ρ, j), H_i represents an execution of $N_1^{\rho,j}$ which terminates in an accepting state, then accept. If, for some (i, ρ, j), H_i represents an execution of $N_2^{\rho,j}$ which terminates in a rejecting state, then reject. Otherwise, continue.

If we are in a "yes" instance, then there exists a certificate vector for which all agents executing N_1 accept, and for any certificate vector, all agents that execute N_2 accept. Therefore, an agent that executes f can never reject and, since by Remark 4 each certificate vector is eventually tested, the agent will eventually accept. Conversely, if we are in a "no" instace, then for any certificate vector, all agents that execute N_1 reject, and there exists a certificate vector for which all agents executing N_2 reject. Therefore, an agent that executes f can never accept and it will eventually reject. The function f is therefore a global decider, and all agents that halt by executing f accept if we are in a "yes" instance, or reject if we are in a "no" instance.

The function g is a local decider for similar reasons, and in fact any agent that halts by executing g accepts in a "yes" instance, or rejects in a "no" instance. Suppose that we are in a "yes" instance and let $\vec{y}^\star$ be the certificate vector that causes all agents executing N_1 to accept. By Remark 4, whatever the particular agent IDs are, there exists a pair (ρ, j) such that, whenever the agents execute $N_1^{\rho,j}$ in their list of protocols, they are actually executing N_1 with the certificate vector $\vec{y}^\star$. Therefore, after a sufficiently large number of phases of the meta-protocol, $N_1^{\rho,j}$ will be eventually fully executed, causing at least one agent to accept as a result of executing g. At the same time, since we are in a "yes" instance, for all (ρ, j), the protocols $N_2^{\rho,j}$ result in agents accepting, therefore no agent can reject as a result of executing g. On the other hand, in a "no" instance, there exists a certificate vector that causes all agents executing N_2 to reject. By similar arguments, it follows that g satisfies the local decider property and, in fact, no agent can accept as a result of executing g. $\square$

Remark 5. Theorem 1 can also be obtained as a corollary of Theorems 3 and 4. Indeed, we know by definition that $\mathsf{MAD} \subseteq \mathsf{co\text{-}MAV'}$ and $\mathsf{co\text{-}MAD} \subseteq \mathsf{MAV'}$, and thus by Theorem 3 we have $\mathsf{MAD} \subseteq \mathsf{co\text{-}MAV_s}$ and $\mathsf{co\text{-}MAD} \subseteq \mathsf{MAV_s}$. It follows, then, that $\mathsf{MAD} \cap \mathsf{co\text{-}MAD} \subseteq \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV_s}$, which yields $\mathsf{MAD} \cap \mathsf{co\text{-}MAD} \subseteq \mathsf{MAD_s}$ by Theorem 4.

Theorem 5. $\mathsf{MAD} = \mathsf{MAV} \cap \mathsf{co\text{-}MAV_s}$ *and* $\mathsf{co\text{-}MAD} = \mathsf{MAV_s} \cap \mathsf{co\text{-}MAV}$.

Proof. It suffices to show that $\mathsf{MAV} \cap \mathsf{co\text{-}MAV_s} \subseteq \mathsf{MAD}$. Let N_1 be a MAV protocol for Π and let N_2 be a $\mathsf{co\text{-}MAV_s}$ protocol for Π, where $\Pi \in \mathsf{MAV} \cap \mathsf{co\text{-}MAV_s}$. In order to show that $\Pi \in \mathsf{MAD}$, we will use the meta-protocol $\mathcal{P}_{\mathcal{N},f,g}$ with the same $\mathcal{N}$, f, and g as in the proof of Theorem 4, except that whenever N_1 is referenced we substitute the MAV protocol for Π.

Suppose that for some implicit input $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$, some agent rejects while executing $\mathcal{P}_{\mathcal{N},f,g}$. If the agent rejects by executing f, then there exists a certificate vector for which all agents executing N_2 on this implicit input reject. Therefore, the implicit input corresponds to a "no" instance of Π. If the agent rejects by executing g, then there exists a certificate vector which causes at least one agent executing N_2 to reject. By the definition of $\mathsf{co\text{-}MAV_s}$, this implies that the implicit input corresponds to a "no" instance.

Now, suppose that all agents accept while executing $\mathcal{P}_{\mathcal{N},f,g}$. If at least one of them accepts by executing f, then it knows the implicit input and it decides correctly, therefore the implicit input corresponds to a "yes" instance. If all of them accept by executing g, then for each agent i let $N_1\big(\mathsf{id}_i, \langle x_i, y_{i,i}\rangle\, ; \vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{y_i}\rangle\big)$ be the accepting execution of N_1 that led that agent to accept. That is, agent i executed one of the protocols $N_1^{\rho_i,j_i}$ for T_i steps and this execution corresponded to an accepting execution of N_1 with certificate vector $\vec{y_i}$ (thus the particular certificate string of agent i was $y_{i,i}$). By Lemma 3, for each agent i, the starting positions of all other agents are at distance strictly greater than $2T_i$ from s_i. Therefore, if agent i executes protocol N_1 with explicit input $\big(\mathsf{id}_i, \langle x_i, y_{i,i}\rangle\big)$, it halts after T_i steps without meeting any other agent and, in fact, we have the stronger property that it will halt after T_i steps whatever certificates are provided to the other agents. Consequently, the execution of N_1 with implicit input $\big(\vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \vec{z}\rangle\big)$, where $\vec{z} = (y_{1,1}, y_{2,2}, \ldots, y_{|\vec{s}|,|\vec{s}|})$, must result in all agents accepting. By the definition of MAV, this implies that $(G, \vec{s}, \vec{x})$ is a "yes" instance of Π. $\square$

Note that it was shown in [19] that, if we consider decision problems that are decidable or verifiable under the promise that the instance contains a single agent (thus giving rise to the classes $\mathsf{MAD_1}$ and $\mathsf{MAV_1}$), then it holds that

$\mathsf{MAD}_1 = \mathsf{MAV}_1 \cap \text{co-}\mathsf{MAV}_1$. Theorems 4 and 5 can be seen as generalizations of that result to multiagent classes.

Proposition 9. $\mathsf{accompanied} \in \mathsf{MAV_s} \setminus \text{co-}\mathsf{MAD}$ *and* $\mathsf{consensus} \in \text{co-}\mathsf{MAV_s} \setminus \mathsf{MAD}$.

Proof. It suffices to show that $\mathsf{accompanied} \notin \text{co-}\mathsf{MAD}$ and $\mathsf{consensus} \notin \mathsf{MAD}$, by Proposition 5.

For a contradiction, let M be a MAD protocol for either $\overline{\mathsf{accompanied}}$ or $\mathsf{consensus}$. An agent that executes M with explicit input $(\mathsf{id}, \varepsilon)$ and implicit input $\big((\mathsf{id}), G, (s), (\varepsilon)\big)$, where G is a ring with an arbitrary port labeling and s is an arbitrary node of G (i.e., the agent is executing M with an empty input, alone on a ring), must accept, say after T_1 steps. Similarly, an agent with explicit input $(\mathsf{id}', 0)$ and implicit input $\big((\mathsf{id}'), G, (s), (0)\big)$, where $\mathsf{id}' \neq \mathsf{id}$, must accept, say after T_2 steps. Now, let G' be the graph obtained by connecting one of the bottom-level nodes of $\mathcal{V}_G^{(T_1+1)}(s)$ (cf. Definition 1) to one of the bottom-level nodes of $\mathcal{V}_G^{(T_2+1)}(s)$, via an edge with port number 2 at both endpoints. Let s_1, s_2 be the nodes of G' that respectively correspond to the roots of the two truncated view graphs that form G'. Consider the execution $M\big((\mathsf{id}, \mathsf{id}'), G', (s_1, s_2), (\varepsilon, 0)\big)$. Since there are several agents, with different inputs, and M is a MAD protocol for either $\overline{\mathsf{accompanied}}$ or $\mathsf{consensus}$, at least one agent should reject. However, by Proposition 1, both of them will accept, which is a contradiction. □

In view of Theorem 5, Proposition 9 yields a separation between $\mathsf{MAV_s}$ and $\text{co-}\mathsf{MAV}$, as $\mathsf{accompanied} \in \mathsf{MAV_s} \setminus \text{co-}\mathsf{MAV}$, and a separation between $\text{co-}\mathsf{MAV_s}$ and MAV, as $\mathsf{consensus} \in \text{co-}\mathsf{MAV_s} \setminus \mathsf{MAV}$.

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 $\text{co-}\mathsf{MAV}$, as illustrated in Figure 2.

6. Closure properties

In this section, we discuss closure properties of the various classes that we have considered in this work under the set-theoretic operations of union, intersection, and complement. Recall that these are summarized in Table 1.

The class $\mathsf{MAD_s}$ is easily seen to be closed under union: If $\Pi_1, \Pi_2 \in \mathsf{MAD_s}$ via protocols M_1 and M_2, respectively, then $\Pi_1 \cup \Pi_2 \in \mathsf{MAD_s}$ via a straightforward application of the meta-protocol for $\mathcal{N} = (M_1, M_2)$. The global decider f just simulates both M_1 and M_2 on the implicit input and accepts if and only if at least one of them results in all agents accepting. The local decider g accepts if either of the histories H_1 and H_2 is accepting, rejects if both are rejecting, and otherwise continues. We have already observed that $\mathsf{MAD_s}$ is closed under complement. Therefore, $\mathsf{MAD_s}$ must be closed under intersection as well.

The class MAD is clearly not closed under complement: If it were, then by Proposition 4 we would get that $\overline{\mathsf{allempty}} \in (\mathsf{MAD} \cap \text{co-}\mathsf{MAD}) \setminus \mathsf{MAD_s}$, which contradicts Theorem 1. Furthermore, MAD is not closed under union: Consider the problems $\mathsf{allempty}$ and $\mathsf{allzero} = \{(G, \vec{s}, \vec{x}) : \forall i \ x_i = 0\}$, which are both in MAD. We have, though, that $\mathsf{allempty} \cup \mathsf{allzero} \notin \mathsf{MAD}$, as we can place two agents with inputs ε and 0 (a "no" instance of $\mathsf{allempty} \cup \mathsf{allzero}$) on antipodal nodes of a sufficiently large ring, which will force them to decide while unaware of each other's existence, therefore both will accept. We can show, however, that MAD is closed under intersection. If $\Pi_1, \Pi_2 \in \mathsf{MAD}$ via protocols M_1 and M_2, respectively, then $\Pi_1 \cap \Pi_2 \in \mathsf{MAD}$ via a straightforward application of the meta-protocol for $\mathcal{N} = (M_1, M_2)$. The global decider f simulates both M_1 and M_2 on the implicit input and accepts if and only if both of them result in all agents accepting. The local decider g accepts if both of the histories H_1 and H_2 are accepting, rejects if at least one is rejecting, and otherwise continues. From the closure properties of MAD, we obtain immediately that $\mathsf{co\text{-}MAD}$ is closed under union and it is not closed under intersection and under complement.

By standard applications of the meta-protocol similar to the ones described before, we can also obtain that $\mathsf{MAV_s}$, and thus $\mathsf{co\text{-}MAV_s}$, are both closed under union and intersection, that MAV is closed under intersection, and thus that $\mathsf{co\text{-}MAV}$ is closed under union. However, $\mathsf{MAV_s}$ and $\mathsf{co\text{-}MAV_s}$ are not closed under complement, because that would yield $\mathsf{accompanied} \in (\mathsf{MAV_s} \cap \mathsf{co\text{-}MAV_s}) \setminus \mathsf{co\text{-}MAD}$, contradicting Theorem 4. MAV and $\mathsf{co\text{-}MAV}$ are also not closed under complement because $\mathsf{degree} \in \mathsf{MAV} \setminus \mathsf{co\text{-}MAV}$. Finally, MAV is not closed under union, which implies that $\mathsf{co\text{-}MAV}$ is not closed under intersection, for the following reasons. Consider the problems degree and $\mathsf{diffdeg} = \{(G, \vec{s}, \vec{x}) : \forall i \ G$ contains at least x_i nodes of pairwise different degrees$\}$, which are both in MAV. We have, though, that $\mathsf{degree} \cup \mathsf{diffdeg} \notin \mathsf{MAV}$. Indeed, an agent with input 1 alone in a ring will accept for some certificate, as this forms a "yes" instance of $\mathsf{diffdeg}$ (although it is a "no" instance of degree). Similarly, an agent with input 2 alone in a ring will accept for some certificate, as this forms a "yes" instance of degree (although it is a "no" instance of $\mathsf{diffdeg}$). Now, placing these agents with inputs 1 and 2 (a "no" instance of $\mathsf{degree} \cup \mathsf{diffdeg}$) on antipodal nodes of a sufficiently large ring with the same certificates will force them to decide while unaware of each other's existence, and therefore both will accept.

7. Relaxing the unanimity constraints

In the classes that we have considered so far, it was always the case that the agents needed to give a unanimous answer either for all "yes" instances or for all "no" instances (or in both cases for $\mathsf{MAD_s}$). One might be tempted to define a "relaxed" version of these classes, in which the only requirement is that at least one agent gives the correct answer, as follows:

Definition 9. *A decision problem Π is in $\mathsf{MAD_r}$ if and only if there exists a mobile agent protocol M such that for all instances $(G, \vec{s}, \vec{x})$: if $(G, \vec{s}, \vec{x}) \in \Pi$ then $\forall \vec{\mathsf{id}}\ M(\vec{\mathsf{id}}, G, \vec{s}, \vec{x}) \mapsto \widehat{\mathsf{yes}}$, whereas if $(G, \vec{s}, \vec{x}) \notin \Pi$ then $\forall \vec{\mathsf{id}}\ M(\vec{\mathsf{id}}, G, \vec{s}, \vec{x}) \mapsto \widehat{\mathsf{no}}$.*

It is easy to check that $\mathsf{MAD_r} = \mathsf{co\text{-}MAD_r}$. Moreover, given the looseness of the acceptance mechanism in $\mathsf{MAD_r}$, it is perhaps not surprising that it contains problems of varying difficulty. Indeed, let us define the following transformations of mobile agent decision problems:

Definition 10. *Let* Π *be any mobile agent decision problem. Then, we define the following problems:*

- $[\Pi]_\varepsilon = \mathsf{allempty} \cup (\mathsf{someempty} \cap \Pi) = \mathsf{someempty} \cap (\mathsf{allempty} \cup \Pi)$
- $[\Pi]_r = \mathsf{accompanied} \cap \mathsf{somenodes\text{-}ub} \cap \Pi$

It turns out that $\mathsf{MAD_r}$ contains both $[\Pi]_\varepsilon$ and $[\Pi]_r$ for any problem Π:

Proposition 10. *For every mobile agent decision problem* Π*,* $[\Pi]_\varepsilon \in \mathsf{MAD_r}$ *and* $[\Pi]_r \in \mathsf{MAD_r}$*.*

Proof. It is easy to verify that the following protocol decides $[\Pi]_\varepsilon$ in the $\mathsf{MAD_r}$ sense: on input x, each agent accepts if $x = \varepsilon$ and rejects otherwise.

For $[\Pi]_r$, consider the following protocol, where Procedure Rdv$(\cdot,\cdot)$ is the procedure that we defined in Section 4.1:

Explicit input: (id_i, x_i). *Implicit input:* $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$.

```
1: Execute Rdv($x_i$, $\mathsf{id}_i$)
2: if met no agents or met some agent with $(x_j, \mathsf{id}_j)$ lexicographically larger
   than $(x_i, \mathsf{id}_i)$ then
3:     reject
4: else
5:     accept
6: end if
```

Observe that the execution of the above protocol on *any* implicit input $(\vec{\mathsf{id}}, G, \vec{s}, \vec{x})$ will result in at least one agent rejecting. Indeed, the agent with the lexicographically smallest pair (x, id) will always reject, whether it meets any other agent or not. It follows, then, that in every "no" instance of $[\Pi]_r$, we have the desired property that at least one agent rejects.

In a "yes" instance of $[\Pi]_r$, there are at least two agents and at least one of them has received an input that is an upper bound for $|V(G)|$. Therefore, the agent with the lexicographically largest (x_i, id_i) will meet every other agent during step 1 and it will accept at line 5. Indeed, consider any other agent, with explicit input (id_j, x_j). If $x_j < x_i$, then $B(x_i) > 2B(x_j)$ and thus the last call to Explore-Ball(x_i) by the agent i will find the halted agent j. Otherwise the two Rdv procedures are synchronized and the meeting will occur before both agents halt. See Section 4.1 for justifications of both cases. □

The class $\mathsf{MAD_r}$, as defined above, can be seen as being too large, in that it contains problems which are arbitrarily high in the arithmetic hierarchy:

Theorem 6. *For every* $j \geq 1$*,* $\mathsf{MAD_r}$ *contains a* Σ^0_j*-complete problem.*

Proof. Fix an encoding of Turing machines as strings over $\Sigma^\star$, where ε encodes a Turing machine that halts immediately without performing any transitions. We show that $[\mathsf{allhalting}_j]_\varepsilon$, which is in $\mathsf{MAD_r}$ by Proposition 10, is $\mathbf{\Sigma}^0_j$-complete for all $j \geq 1$. The problem $\mathsf{allhalting}_j$ is easily seen to be in $\mathbf{\Sigma}^0_j$, therefore also $[\mathsf{allhalting}_j]_\varepsilon \in \mathbf{\Sigma}^0_j$.

Moreover, $\mathsf{HP}_j \leq_m [\mathsf{allhalting}_j]_\varepsilon$ via the following reduction: On input x, the reduction returns $(G, \vec{s}, \vec{x})$ where G is a graph with one node and $\vec{x} = (\varepsilon, x)$. Indeed, if $x \in \mathsf{HP}_j$, then $(G, \vec{s}, \vec{x}) \in [\mathsf{allhalting}_j]_\varepsilon$ because at least one agent receives ε and both inputs correspond to Turing machines with oracle HP_{j-1} that halt when executed with input ε. Conversely, if $(G, \vec{s}, \vec{x}) \in [\mathsf{allhalting}_j]_\varepsilon$, then either $(G, \vec{s}, \vec{x}) \in \mathsf{allempty}$ and therefore $x = \varepsilon \in \mathsf{HP}_j$, or $(G, \vec{s}, \vec{x}) \in \mathsf{allhalting}_j$ and therefore $x \in \mathsf{HP}_j$. □

On the other hand, there are also problems that are arbitrarily high in the arithmetic hierarchy and do not belong to $\mathsf{MAD_r}$, like $\mathsf{allhalting}_j$, which is $\mathbf{\Sigma}^0_j$-complete:

Proposition 11. *For all $j \geq 1$, $\mathsf{allhalting}_j \notin \mathsf{MAD_r}$.*

Proof. If $\mathsf{allhalting}_j \in \mathsf{MAD_r}$, then this would imply that there exists a protocol that decides whether a single agent on a single-node graph has received an input $x \in \mathsf{HP}_j$. However, HP_j is undecidable. □

In fact, even some problems in $\mathbf{\Delta}^0_1$ do not belong to $\mathsf{MAD_r}$. This is for example the case for the problem path, again because intuitively a single agent cannot distinguish a (sufficiently long) path from a cycle. We end this section by presenting some additional results linking the standard computability classes and the mobile agent computability classes thanks to some properties of $\mathsf{MAD_r}$.

Theorem 7. $\mathsf{MAD_r} \cap \mathbf{\Sigma}^0_1 \subseteq \mathsf{MAV}$.

Proof. Let M be a $\mathsf{MAD_r}$ protocol for a mobile agent decision problem Π, and assume that Π is also in $\mathbf{\Sigma}^0_1$. Recall that $\Pi \in \mathbf{\Sigma}^0_1$ means that there exists a decidable predicate $R(\cdot, \cdot)$ such that, for every instance I encoded by $\langle I \rangle \in \Sigma^\star$, $I \in \Pi \Leftrightarrow \exists c \in \Sigma^\star\, R\big(\langle I \rangle, c\big)$.

We will use the meta-protocol $\mathcal{P}_{\mathcal{N},f,g}$ with explicit input of the form $\big(\mathsf{id}, \langle x, \langle y, c \rangle \rangle\big)$, in order to give a MAV protocol for Π. Let $\mathcal{N} = (M)$ and f, g be defined as follows: To compute $f\big(\mathsf{id}, \langle x, \langle y, c \rangle \rangle\, ; \vec{\mathsf{id}}, G, \vec{s}, \langle \vec{x}, \langle \vec{y}, \vec{c} \rangle \rangle\big)$, return **accept** if $R\big(\langle I \rangle, c\big)$, where $\langle I \rangle$ is the encoding of the instance $I = (G, \vec{s}, \langle \vec{x}, \langle \vec{y}, \vec{c} \rangle \rangle)$, otherwise return **reject**. To compute $g(T, \mathsf{id}, \langle x, \langle y, c \rangle \rangle, H)$, if $T < y$ or H does not terminate, return **continue**. Otherwise, return **accept** if H accepts, or **reject** if H rejects.

Observe that, in a "no" instance, for every certificate vector $\langle \vec{y}, \vec{c} \rangle$, at least one agent rejects. Indeed, if all of the agents decide via the local decider g, then by Corollary 1 and the definition of the $\mathsf{MAD_r}$ protocol for Π, at least one of them will reject. If at least one decides via the global decider f, then that agent will reject by definition of the predicate R.

In a "yes" instance I, we can provide to the agents the certificate vector $\langle \vec{n}, \vec{c} \rangle$, where $\vec{n}$ is a vector with all components equal to the size n of the graph, and $\vec{c}$ is a vector with all components equal to the string c which renders $R\big(\langle I \rangle, c\big)$ true. With this certificate, if some agent decides during phase T via the local decider, then this means that $T \geq n$ and, by Lemma 3, that agent is the only agent in the instance and therefore it will accept by definition of the $\mathsf{MAD_r}$ protocol. Consequently, if there are two or more agents, they will all decide via the global decider f, and therefore they will all evaluate $R\big(\langle I \rangle, c\big)$ and accept. In all cases, given the certificate vector $\langle \vec{n}, \vec{c} \rangle$, all agents accept. □

Corollary 2. $\mathsf{MAD_r} \cap \mathsf{\Pi}^0_1 \subseteq \text{co-}\mathsf{MAV}$.

Proof. This follows from Theorem 7 and the fact that $\mathsf{MAD_r} = \text{co-}\mathsf{MAD_r}$. □

Corollary 3. $\mathsf{MAD_r} \cap \mathsf{\Delta}^0_1 \subseteq \mathsf{MAV} \cap \text{co-}\mathsf{MAV}$.

Proof. This follows from Theorem 7, Corollary 2, and the fact that $\mathsf{\Delta}^0_1 = \mathsf{\Sigma}^0_1 \cap \mathsf{\Pi}^0_1$. □

Corollary 4. *For every mobile agent decision problem* $\Pi \in \mathsf{\Sigma}^0_1$*,* $[\Pi]_\varepsilon \in \mathsf{MAV}$ *and* $[\Pi]_r \in \mathsf{MAV}$*.*

Proof. This follows from Proposition 10, Theorem 7, and the observation that if $\Pi \in \mathsf{\Sigma}^0_1$, then clearly also $[\Pi]_\varepsilon \in \mathsf{\Sigma}^0_1$ and $[\Pi]_r \in \mathsf{\Sigma}^0_1$. □

Corollary 5. *For every mobile agent decision problem* $\Pi \in \mathsf{\Pi}^0_1$*,* $[\Pi]_\varepsilon \in \text{co-}\mathsf{MAV}$ *and* $[\Pi]_r \in \text{co-}\mathsf{MAV}$*.*

Proof. This follows from Proposition 10, Corollary 2, and the observation that if $\Pi \in \mathsf{\Pi}^0_1$, then clearly also $[\Pi]_\varepsilon \in \mathsf{\Pi}^0_1$ and $[\Pi]_r \in \mathsf{\Pi}^0_1$. □

8. Discussion

A common feature of the classes that we considered in this work is that the certificates that are given to the agents as part of the verification protocol do not depend on the agent identifiers. This was also the case in [19]. This is consistent with the fundamental assumption that the property to be decided by the system of mobile agents involves only the graph, the starting positions of the agents, and their respective inputs (cf. Definition 3). In other words, the membership of a given instance in a given mobile agent decision problem is independent of the assignment of identifiers to the agents, which can be seen as saying that the agent identifiers are essentially only used for symmetry breaking purposes. Nevertheless, it would also be interesting to consider mobile agent verification classes in which the certificates may depend on the agent identifiers.

A further noteworthy point is that, apparently, the main source of difficulty for the decidability of mobile agent problems is the fact that the agents cannot distinguish between instances that contain only one agent and instances that contain two or more agents. Indeed, under the promise that the instance contains a single agent, the corresponding classes satisfy $\mathsf{MAD}_1 = \mathsf{co\text{-}MAD}_1 = \mathsf{MAV}_1 \cap \mathsf{co\text{-}MAV}_1$ [19] and thus the inclusion diagram of Figure 3 collapses to a great extent. On the other hand, under the promise that there are two or more agents, the agents can perform rendezvous with increasing guesses on the number of nodes until they learn the implicit input, and thus they decide all decidable problems. Conversely, a promise that the instance contains at most two agents does not seem to give any immediately exploitable information to a verification algorithm. It would be interesting if one could formalize this observation by comparing MAV to the class of problems that admit a MAV protocol under the promise that the instance contains at most two agents.

We leave as an interesting and challenging direction for future work the computability-theoretic classification of mobile agent problems that do not fall within the framework of decision problems, such as exploration, map construction, rendezvous or pattern formation problems, among others. This may require significant adaptation of the current class definitions and techniques.

Acknowledgment

We are grateful to two anonymous referees for their careful reading of the manuscript and their numerous and pertinent suggestions and remarks.

This work was partially funded by the ANR projects MACARON (ANR-13-JS02-002) and DESCARTES (ANR-16-CE40-0023). This study has been carried out in the frame of the "Investments for the future" Programme IdEx Bordeaux – CPU (ANR-10-IDEX-03-02).

[1] Ambühl, C., Gasieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. ACM Trans. Algorithms 7(2), 17:1–17:21 (2011)

[2] Blin, L., Fraigniaud, P., Nisse, N., Vial, S.: Distributed chasing of network intruders. Theor. Comput. Sci. 399(1-2), 12–37 (2008)

[3] Boldi, P., Vigna, S.: An effective characterization of computability in anonymous networks. In: DISC 2001. LNCS, vol. 2180, pp. 33–47. Springer (2001)

[4] Boldi, P., Vigna, S.: Universal dynamic synchronous self-stabilization. Distrib. Comput. 15(3), 137–153 (2002)

[5] Censor-Hillel, K., Fischer, E., Schwartzman, G., Vasudev, Y.: Fast Distributed Algorithms for Testing Graph Properties. In: DISC 2016. LNCS, vol. 9888, pp. 43–56. Springer (2016)

[6] 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)

[7] 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)

[8] Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM 43(2), 225–267 (1996)

[9] Das, S.: Mobile agents in distributed computing: Network exploration. Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 109, 54–69 (2013)

[10] Das S., Kutten S., Lotker Z.: Distributed verification using mobile agents. In: ICDCN 2013. LNCS, vol 7730, pp. 330–347. Springer (2013)

[11] Das Sarma, A., Holzer, S., Kor, L., Korman, A., Nanongkai, D., Pandurangan, G., Peleg, D., Wattenhofer, R.: Distributed Verification and Hardness of Distributed Approximation. SIAM J. Comput. 41(5), 1235–1265, (2012)

[12] Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Searching for a black hole in arbitrary networks: optimal mobile agents protocols. Distributed Computing 19(1), 1–18 (2006)

[13] Emek, Y., Pfister, C., Seidel, J. Wattenhofer, R.: Anonymous networks: randomization = 2-hop coloring. In: PODC 2014. pp. 96–105. ACM (2014)

[14] Feuilloley, L., Fraigniaud, P.: Survey of Distributed Decision. Bulletin of the EATCS 119 (2016)

[15] Fomin, F. V., Thilikos, D. M.: An annotated bibliography on guaranteed graph searching. Theor. Comput. Sci. 399(3), 236–245 (2008)

[16] Fraigniaud, P., Göös, M., Korman, A., Suomela, J.: What can be decided locally without identifiers? In: PODC 2013. pp. 157–165. ACM (2013)

[17] 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)

[18] Fraigniaud, P., Korman, A., Peleg, D.: Towards a complexity theory for local distributed computing. J. ACM 60(5), 35 (2013)

[19] Fraigniaud, P., Pelc, A.: Decidability classes for mobile agents computing. J. Parallel Distrib. Comput. 109, 117–128 (2017)

[20] Fraigniaud, P., Rajsbaum, S., and Travers, C.: Locality and checkability in wait-free computing. Distrib. Comp. 26(4), 223–242, (2013)

[21] Göös, M., Suomela, J.: Locally Checkable Proofs in Distributed Computing. Theory of Computing 12(1), 1–33, (2016)

[22] Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. 13(1), 124–149 (1991)

[23] Korman, A., Kutten, S., Peleg, D.: Proof labeling schemes. Distrib. Comp. 22(4), 215–233, (2010)

[24] Lange, D.B., Oshima, M.: Seven good reasons for mobile agents. Commun. ACM 42(3), 88–89 (1999)

[25] Markou, E.: Identifying hostile nodes in networks using mobile agents. Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 108, 93–129 (2012)

[26] Moni Naor, M., Stockmeyer, L. J.: What can be computed locally? SIAM J. Comput. 24(6), 1259–1277, (1995)

[27] Pelc, A.: Deterministic rendezvous in networks: A comprehensive survey. Networks 59(3), 331–347 (2012)

[28] Yamashita, M., Kameda, T.: Computing functions on asynchronous anonymous networks. Math. Syst. Theory 29(4), 331–356 (1996)