Computability in Anonymous Networks: Revocable vs. Irrecovable Outputs[⋆]

Yuval Emek[1], Jochen Seidel[2], and Roger Wattenhofer[2]

[1] Faculty of Industrial Engineering and Management, Technion, Haifa, Israel
`yemek@ie.technion.ac.il`
[2] Distributed Computing, ETH Zürich, Zürich, Switzerland
{`seidelj, wattenhofer`}`@ethz.ch`

Abstract. What can be computed in an anonymous network, where nodes are not equipped with unique identifiers? It turns out that the answer to this question depends on the commitment of the nodes to their first computed output value: Two classes of problems solvable in anonymous networks are defined, where in the first class nodes are allowed to revoke their outputs and in the second class they are not. These two classes are then related to the class of all centrally solvable network problems, observing that the three classes form a strict linear hierarchy, and for several classic and/or characteristic problems in distributed computing, we determine the exact class to which they belong.
Does this hierarchy exhibit complete problems? We answer this question in the affirmative by introducing the concept of a distributed oracle, thus establishing a more fine grained classification for distributed computability which we apply to the classic/characteristic problems. Among our findings is the observation that the three classes are characterized by the three pillars of distributed computing, namely, local symmetry breaking, coordination, and leader election.

1 Introduction

We study computability in networks, referred to hereafter as *distributed computability*. Distributed computability is equivalent to classic centralized (Turing Machine) computability when the nodes are equipped with unique (comparable) identifiers. However, as Angluin noticed in her seminal work [3], distributed computability becomes fascinating in *anonymous* networks, where nodes do not have unique IDs. What can be computed with deterministic algorithms merely depends on the topology of the network, and it is well known that problems like maximal independent set can be solved in an anonymous network only if the nodes are allowed to toss coins. We therefore consider the distributed computability of *randomized* algorithms running in anonymous networks. Notice that

[⋆] Due to space limitations most proofs are omitted or replaced by proof sketches in this extended abstract. Also most results obtained in Section 4 are left out. We refer the interested reader to the full version which is available at `http://disco.ethz.ch/publications/ICALP2014-revocability-full.pdf`.

in the scope of this paper, we do not impose any limitations on the complexity resources (time, message/memory size, ...), however, like in classic sequential computability theory, we do require a correct result after a finite amount of time.

Apart from its theoretical interest, the study of anonymous networks is motivated by various real-world scenarios. For example, the nodes may be indistinguishable due to their fabrication in a large-scale industrial process [5], in which equipping every node with a unique identifier (serial number) is not economically feasible. In other cases nodes may not wish to reveal their unique identity out of privacy and security concerns [24].

1.1 Setting

Distributed Problems. We consider simple (undirected, loop-free and no parallel edges) connected finite graphs G, and denote the node and edge sets of a graph G by $V(G)$ and $E(G)$ or V and E if G is clear from the context. A function $f : V(G) \to L$ is called a *labeling* of the graph G, and we refer to the set L as the set of *values* that f assigns to nodes in G. A *distributed problem Π* is a set of three-tuples (G, i, o), where G is a graph as described above, and i and o are *input labels* and *output labels* for G. For every problem there are two sets $I(\Pi)$ and $O(\Pi)$ denoting the *input values* and *output values* of Π, i.e., the values that the labels i and o assign, correspondingly. Such a three-tuple $(G, i, o) \in \Pi$ is called a *(solved) instance* of Π. An *input instance* of Π is a two-tuple (G, i) for which there exists a *valid output* o satisfying $(G, i, o) \in \Pi$, and we also write $(G, i) \in \Pi$ for input instances of the problem. We restrict ourselves to problems that are solvable in a centralized setting.

Randomized Anonymous Algorithms. Our definition of how distributed algorithms work follows the convention of [30] for synchronized network systems (message passing) with simultaneous starting times. Nodes execute the same randomized and uniform algorithm in synchronous *rounds*, and in each round we allow each node access to finitely many random bits. Every node v knows its degree $\deg(v)$ and can distinguish between its neighbors $\Gamma(v)$ (by means of a bijection $\{1, \ldots, \deg(v)\} \to \Gamma(v)$, cf. the *port model*). In each round every node sends and receives a message of unbounded, yet finite, size to and from each individual neighbor. To ease our discussion every node v is equipped with one *input register* holding some problem-dependent input value and one *output register*. The output register initially contains a special symbol ε indicating *v is not ready* to return an output. Any value $x \neq \varepsilon$ contained in v's output register is interpreted as v being *ready* to return its output and we say that *v has output x*. A global *configuration* in which all nodes are ready is called a *ready configuration*. When algorithm $\mathcal{A}$ is in a ready configuration, we define *$\mathcal{A}$'s output* $o_{\mathcal{A}} : V(G) \to O$ by setting $o_{\mathcal{A}}(v)$ to be the content of node v's output register. In the following, we consider two different notions of *output revocability*.

Definition (Output Revocability). An algorithm is referred to as a *write-once* algorithm if every node is restricted to write to its output register at most once. If this restriction is lifted, then we call it a *rewrite* algorithm.

In other words, in a rewrite algorithm a node may revoke its output, e.g., by writing ε to its output register. While every execution of a write-once algorithm reaches at most one ready configuration, during the execution of a rewrite algorithm many ready configurations can occur. Note that the converse does not hold: an algorithm that is guaranteed to reach at most one ready configuration is not necessarily a write-once algorithm. In the existing literature, algorithms are typically considered to be write-once algorithms.

Definition (Correctness). Fix some problem Π and an algorithm $\mathcal{A}$. A ready configuration of $\mathcal{A}$ when invoked on an input instance $(G, i) \in \Pi$ is said to be *valid* if the output $o_{\mathcal{A}}$ of $\mathcal{A}$ in this configuration is a valid output for (G, i). Algorithm $\mathcal{A}$ is said to *solve* Π if it satisfies the following two conditions for every input instance $(G, i) \in \Pi$: (1) A ready configuration is reached within finite time with probability 1. (2) Every ready configuration that can occur with a positive probability is valid.

The aforementioned definition of correctness requires that all occurring ready configurations will be correct (i.e., correspond to a valid output). In Section 2 we show that our definition of correctness is robust to certain changes. Notice that in the scope of this paper, we do not require that an algorithm *terminates* in order to be correct. However, the algorithms designed throughout the paper do terminate, and the general transformation techniques we present (i.e., compilers/simulations) can be designed to ensure termination if the algorithms to which the transformation is applied terminate.

The choice of output revocability has a significant impact on the problems that an algorithm can solve. In the following the terms WO-algorithms and RW-algorithms will thus be used to denominate write-once and rewrite algorithms running in an anonymous network, respectively; RW and WO refer to the *classes* of distributed problems solvable by these two types of algorithms. Lastly, we denote by CF the class of distributed problems that are solvable in a centralized setting (by a Turing machine), bearing in mind that this class essentially includes every *computable function* on graphs. The distinction of these classes is justified by the following observation. The full version of this paper contains a straightforward proof.

Observation. *The classes of distributed problems satisfy* WO $\subset$ RW $\subset$ CF *(in the strict sense).*

1.2 Our Contribution

What can be computed in anonymous networks? As it turns out the effect output revocability has on the distributed computability of anonymous networks is remarkable. A total of 21 problems, including some of the most fundamental problems in distributed computing, are classified according to the exact class to which they belong (Section 4).

Does the hierarchy we present exhibit complete problems? To answer this question we introduce the notion of accessing an oracle in a distributed setting and show that this notion is sound (Section 3). As the first stepping stone in this effort we show that the classes WO and RW are robust against two modifications to the aforementioned correctness condition (Section 2). In the full version of this paper each of our 21 problems is then classified according to its *hardness* or *completeness* for the three classes, thus obtaining a deeper understanding of the intrinsic properties of these problems. For reasons of brevity this extended abstract only gives a brief overview of those results (Section 4). Surprisingly, the WO, RW, and CF classes turn out to capture exactly the three *pillars* of distributed computing, namely, local symmetry breaking, coordination, and leader election, respectively.

1.3 Related Work

The history of distributed computability starts with the work of Angluin [3] proving that randomization does not help to elect a leader in anonymous networks. Later, it was shown that electing a leader in an anonymous ring network is possible if the size n of the ring is known [25], in fact, a $(2-\varepsilon)$-approximation of n is enough [1], not only in the special case of a ring but in general networks [34]. It turns out that all these results, and many similar ones, come almost for free once our characterization for the class RW (established in the full version) is available.

There is a line of work that concentrates on *deterministic* distributed algorithms for problems in CF, in particular if some parameters of the topology of the graph (e.g., its size) are known, e.g. [35,10]. Deterministic algorithms are interesting to investigate even if the graph is restricted to a ring [18,13], and also assignments of not necessarily unique identifiers were studied in this context [31].

Another line of research studies computability in anonymous (directed) networks in connection with termination. Not unlike us it is argued that termination in distributed systems is an issue that is not directly evident, since one may be interested in systems where nodes terminate independently of others. The strongest anonymous model considered in [11] is equivalent to deterministic write-once algorithms with knowledge of an upper bound to the network size. When no prior knowledge is assumed the class of solvable problems can be fully characterized using *local views*[3] and recursive functions [14]. Extending their approach, in the current paper an individual node executing a RW-algorithm can never be entirely sure about termination. We show that the class RW lies *between* the two classes WO (local termination) and CF (global termination).

Output revocability should not be confused with the concept of *eventual correctness*, where the network eventually converges to a correct output. For example, *self-stabilizing algorithms* [15] allow the system to return an incorrect output for a finite amount of time, thus allowing a fault-tolerant algorithm to recover from errors. With randomization, self-stabilizing leader election is possible on general graphs [16], hence with randomization every CF-problem is eventually solvable in an anonymous network. In our terminology eventual correctness

[3] Local views are only discussed in the full version of this paper.

could be viewed as requiring that some ready configuration, not necessarily the first one, is stable[4] and valid. We require though that every output returned by the network is correct, but we do allow the network to revoke partial outputs. The problems solvable by self-stabilizing algorithms in directed graphs can be characterized by *fibrations* [12] similar to our characterization for RW that is presented in the full paper. The notion of eventual correctness is also used in the scope of population protocols [5] in which nodes are modeled by finite state machines, see [9] for an overview. In a clique network, the predicates a population protocol can solve are exactly those expressible in first-order Presburger arithmetic [5,6,7], whereas in bounded-degree graphs a Turing machine with linearly bounded space can be simulated [4]. It was also studied how the correctness condition for population protocols affects solvability of the Consensus problem [8].

Apart from these results, not much is known about distributed computability (in contrast to distributed complexity). However, there are surprising connections between complexity and computability, which go beyond us borrowing the terms hardness and completeness. Regarding network algorithms, in the last 30 years, a lot of research went into the question how fast a particular problem can be computed by the network.

Naor and Stockmeyer [33] introduced the notion of locally checkable labelings in identified networks and ask how a constant-time deterministic algorithm can decide whether the labeling represents a correct solution to a given problem. Follow-up work looked at the bit complexity required to solve problems [26,21] and a problem hierarchy depending on the size of checkable labelings was suggested [23], also for anonymous networks. Our work also yields a characterization for the decision problems in RW. However, we do not restrict the run-time to be constant and allow randomization for symmetry breaking. Pruning algorithms [27] were inspired by the same line of research, in an effort to remove the necessity of global knowledge about the graph. While our algorithms are required to give a correct output in every execution [19,22] study the notion of (p,q)-decidable decision problems (an anonymous randomized algorithm may return a wrong output with constant probability) and find a hierarchy among the solvable problem-classes depending on the success probabilities. If a randomized algorithm is allowed to fail (Monte-Carlo algorithm), then a leader can be elected [32] with high probability (w.h.p, i.e., with probability $1 - n^{-c}$ for any c). Hence any CF-problem can be solved in an anonymous network w.h.p. In contrast to that, we require a correct output with probability 1.

Non-deterministic algorithms running in an anonymous setting can fully determine the structure of the radius t-ball around itself in [20], and thus solve exactly the decision problems that are closed under so-called *t-homomorphisms*. In our model only the local view can be retrieved. It may thus be surprising that RW-algorithms can solve exactly the problems that such non-deterministic constant-time algorithms can solve in a single round.

[4] A configuration is said to be *stable* if the nodes no longer revoke their outputs, see Section 2.

2 Notions of Correctness

Our definition of a correct algorithm requires every ready configuration that occurs throughout an execution to be valid. For WO-algorithms this requirement is superfluous since its execution will reach at most one ready configuration. However, RW-algorithms may invalidate or change a ready configuration after it occurred. One may therefore wonder if strengthening the definition by allowing only one durable ready configuration makes the class of solvable problems strictly smaller. On the other hand one may be tempted to weaken this definition, in hope to capture a larger class of problems by requiring only the first occurring ready configuration to be correct. Perhaps surprisingly we show that these two variants have no effect and are equivalent to the current definition of correctness. This equivalence will play a key role when we reason about RW-algorithms in the next section which covers distributed oracles.

Definition (Sustainable Correctness). A ready configuration is said to be *stable*, if the nodes no longer revoke their outputs. Algorithm $\mathcal{A}$ is said to *sustainably solve* a problem Π if it satisfies the following two conditions for every input instance $(G, i) \in \Pi$: (1) A ready configuration is reached within finite time with probability 1. (2) The first ready configuration that occurs is valid and stable.

Definition (Loose Correctness). Algorithm $\mathcal{A}$ is said to *loosely solve* a problem Π if it satisfies the following two conditions for every input instance $(G, i) \in \Pi$: (1) A ready configuration is reached within finite time with probability 1. (2) The first ready configuration that occurs is valid.

The class *Sustainable-RW* (respectively, *Loose-RW*) consists of every distributed problem that can be sustainably solved (resp., loosely solved) by a RW-algorithm. Since sustainable correctness (resp., loose correctness) is a restriction (resp., a relaxation) of correctness as defined in Section 1.1, we conclude that Sustainable-RW $\subseteq$ RW $\subseteq$ Loose-RW. Note that the corresponding classes *Sustainable-WO* and *Loose-WO* for WO-algorithms are equal to the class WO due to the write-once restriction of these algorithms. The following theorem states that also for RW-algorithms the three classes are, in fact, equal.

Theorem 1. *The classes of problems solvable by* RW*-algorithms under the three different notions of correctness satisfy Sustainable-*RW = RW = *Loose-*RW.

The proof of Theorem 1 relies on a *sustainability compiler* that takes a RW-algorithm $\mathcal{A}$ that loosely solves a problem Π and transforms it into a RW-algorithm $\hat{\mathcal{A}}$ that sustainably solves this problem. At the heart of the compiler lies the concept of *inhibiting messages*, i.e., a refinement of a simple concept referred to as *safe broadcast* in which information is broadcast throughout the whole network and no ready configuration is reached before all nodes have received the information. Specifically, the inhibiting messages ensure that the first ready configuration reached by algorithm $\hat{\mathcal{A}}$ is stable. We refer to the appended full version of this paper for the details of the sustainability compiler and its underlying inhibiting message technique.

3 Distributed Oracles

In this section, we introduce the concepts of hardness and completeness, which are central to this work and allow us to gain a deeper understanding how the computability classes relate to each other. To that end, we introduce the notion of an oracle working in a distributed setting.

Definition (Algorithm with access to a $\mathit{\Pi}$-oracle). Consider some problem $\mathit{\Pi}$. A C-algorithm, $\mathrm{C} \in \{\mathrm{WO}, \mathrm{RW}\}$, with *access to a $\mathit{\Pi}$-oracle* is a distributed C-algorithm in which every node v is equipped with a designated *oracle input register* and a designated *oracle output register*. Given some $r \geq 1$, let $\tilde{i}(v)$ be the content of v's oracle input register in round r and let $\tilde{o}(v)$ be the content of v's oracle output register in round $r+1$. If $(G, \tilde{i})$ is an input instance of $\mathit{\Pi}$, then it is guaranteed that $\tilde{o}$ is a valid output for $(G, \tilde{i})$. No assumptions are made on the operation of the algorithm if $(G, \tilde{i}) \notin \mathit{\Pi}$.

While applying the oracle in every round of the algorithm may seem powerful, allowing the distributed algorithm to arbitrarily choose the rounds in which the oracle is applied may require some sort of global coordination, which is not necessarily possible. In comparison, a weaker definition of "accessing an oracle" would be to allow application of the oracle only once in round 1. This distinction does not make a difference for problems $\mathit{\Pi}$ without inputs ($|I(\mathit{\Pi})| = 1$), e.g., for graph theoretic problems like coloring, maximal independent set, or determining the diameter, because the oracle is always applied on the same input instance. It does however affect problems that do receive inputs ($|I(\mathit{\Pi})| \geq 2$), e.g., Consensus or logical And and Or.

As stated above, based on the oracle concept, we will soon introduce the notion of hard and complete problems for the hierarchy of problem classes. This notion would be ill-defined if accessing an oracle to a problem $\mathit{\Pi}_{\mathrm{C}} \in \mathrm{C}$ could enhance the computational power of a C-algorithm. We ensure that the notion of an algorithm with access to an oracle is sound in the following theorem. Note that the statement of the theorem does not mention the case $\mathrm{C} = \mathrm{CF}$, since the soundness of oracles for centralized models is well understood and in any case, beyond the scope of the current paper.

Theorem 3 (Soundness). *If a problem $\mathit{\Pi}$ is solvable by a* C*-algorithm,* $\mathrm{C} \in \{\mathrm{RW}, \mathrm{WO}\}$*, accessing an oracle to a problem $\mathit{\Pi}_{\mathrm{C}} \in \mathrm{C}$, then $\mathit{\Pi}$ can also be solved by a* C*-algorithm that does not access any oracle.*

The key to proving this theorem is to show that in a C-algorithm $\mathcal{A}^{\mathrm{a}}$ that solves a problem $\mathit{\Pi}$ with *access* to a $\mathit{\Pi}_{\mathrm{C}}$-oracle, $\mathit{\Pi}_{\mathrm{C}} \in \mathrm{C}$, one can *replace* the oracle access by simulating a C-algorithm $\mathcal{A}^{\mathrm{r}}$ that solves $\mathit{\Pi}_{\mathrm{C}}$ without any oracle access. This turns out to be a non-trivial task especially for RW-algorithms since a node v simulating $\mathcal{A}^{\mathrm{r}}$ cannot know for sure that the output returned by $\mathcal{A}^{\mathrm{r}}$ will not be revoked later on, i.e., whether it can be safely used for the execution of $\mathcal{A}^{\mathrm{a}}$. In other words, node v does not know when such a result is valid so that the execution of $\mathcal{A}^{\mathrm{a}}$ can continue based on this result (as if it was returned by the

Π_{C}-oracle). The technique we present to resolve this issue for RW-algorithms is based on Theorem 1. Since the sustainability compiler (discussed in detail in the full paper) works independently of the algorithm's access to an oracle, the arguments to establish Theorem 1 can be repeated to yield the following.

Lemma 4. *Fix some problem Π'. Let $\mathcal{A}$ be a* RW*-algorithm with access to a Π'-oracle loosely solving a problem Π and let $\hat{\mathcal{A}}$ be the* RW*-algorithm with access to a Π'-oracle obtained by applying the sustainability compiler to $\mathcal{A}$. Then $\hat{\mathcal{A}}$ sustainably solves Π with an access to a Π'-oracle.*

The ability to transform any RW-algorithm to ensure sustainable correctness plays a key role in the proof of Theorem 3. Recall that our goal is to replace the access to a Π_{C}-oracle of a C-algorithm $\mathcal{A}^{\mathrm{a}}$ by a C-algorithm $\mathcal{A}^{\mathrm{r}}$ solving Π_{C} without any oracle access. In other words, the crux is to show how a C-algorithm $\mathcal{A}$ can interleave the execution of algorithm $\mathcal{A}^{\mathrm{a}}$ with an invocation of $\mathcal{A}^{\mathrm{r}}$ in every round in a correct manner, without any additional knowledge of the run-time of $\mathcal{A}^{\mathrm{r}}$ or properties of the underlying network. As noted before, in the case C = RW, algorithm $\mathcal{A}$ faces the issue that an output returned to a node v by $\mathcal{A}^{\mathrm{r}}$ may not be part of a ready configuration and thus it is not clear whether v should use this value as an output of the Π_{C}-oracle that $\mathcal{A}^{\mathrm{a}}$ invoked. Theorem 1 however relieves $\mathcal{A}$ from the burden of dealing with more than one ready configuration of $\mathcal{A}^{\mathrm{r}}$, whereas Lemma 4 does the same with $\mathcal{A}^{\mathrm{a}}$. Therefore, $\mathcal{A}$ is left with the task of determining when $\mathcal{A}^{\mathrm{r}}$ and $\mathcal{A}^{\mathrm{a}}$ have reached a ready configuration.

In the full version of this paper we show how this can be accomplished by carefully dividing the simulation into phases of a predetermined length and recycling previously used random bits. Assuming that Theorem 3 is established we introduce the concept of hard problems by borrowing the terminology from sequential complexity theory.

Definition (Hardness). For two classes B $\supseteq$ C, a problem Π is said to be B-*hard* with respect to C, denoted by $\Pi \in$ *B-hard*$_{\mathrm{C}}$, if for every problem $\Pi_{\mathrm{B}} \in$ B, there exists a C-algorithm that solves Π_{B} with access to a Π-oracle. We say that Π is *complete* in B with respect to C, denoted by $\Pi \in$ *B-complete*$_{\mathrm{C}}$, if additionally Π itself is contained in B.

Following our notational convention, we would refer to an $\mathcal{NP}$-hard problem as being $\mathcal{NP}$-*hard*$_{\mathcal{P}}$. For example, the problem of electing a leader is well known to be CF-*hard*$_{\mathrm{WO}}$ since once a leader is available, this leader can assign unique identifiers to all other nodes and solve the problem centrally. Our definition yields the three hardness classes CF-*hard*$_{\mathrm{RW}}$, CF-*hard*$_{\mathrm{WO}}$ and RW-*hard*$_{\mathrm{WO}}$, allowing us to study how algorithms running in anonymous networks relate to centralized algorithms as well as how the two output revocability notions relate among each other. By definition, every CF-*hard*$_{\mathrm{WO}}$ problem is both CF-*hard*$_{\mathrm{RW}}$ and RW-*hard*$_{\mathrm{WO}}$; in Section 5 we present a proof sketch for the following theorem, which states that the converse direction is also true. A thorough proof appears in the full version.

Theorem 7. *It holds that* CF-*hard*$_{\mathrm{WO}}$ = CF-*hard*$_{\mathrm{RW}}$ $\cap$ RW-*hard*$_{\mathrm{WO}}$.

4 Problem Zoo

We study the computability and hardness of 21 problems in our setting, and develop different proof techniques to tackle this tedious task. In this extended abstract we confine ourselves to summarize the fruits of our effort in Figure 1. Exemplarily we also present the hardness result for logical OR which is necessary for the sketched proof of Theorem 7 in Section 5.

Overview of Problems. We briefly explain the problems listed in Figure 1.

- LEADER-ELECTION: all but one node output "NOT LEADER", while a single node outputs "LEADER".
- UNIQUENESS: determine whether all nodes have a unique input value.
- IDS: without any input, every node must return a unique identifier.
- α-SIZE-APX: determine a value $\tilde{n}$ such that $n \leq \tilde{n} \leq \alpha \cdot n$, where n is the number of nodes in the network.
- MIN-CUT: determine a partition of the network inducing a minimum cut as well as the size of this cut.
- MIN-CUT-VALUE: determine the size of a minimum cut.
- MIN-CUT-PARTITION: determine a partition of the network inducing a minimum cut.
- DIAMETER: determine the diameter D of the network.
- α-DIAMETER-APX: determine a value $\tilde{d}$ such that $D \leq \tilde{d} \leq \alpha \cdot D$.
- MIN-COLORING: color the graph with the minimum number of colors.
- COLORING: determine *some* coloring of the graph.
- k-HOP-COLORING: color the graph so that the color of every node v differs from the color of every other node in its k-hop neighborhood.
- MIS: determine a maximal independent set.
- k-HOP-MIS: find a maximal subset S of the nodes so that the distance between every two nodes in S is greater than k.
- CONSENSUS: nodes return the same value x which is at least one node's input.
- COORDINATION: determine whether all nodes have the same input.

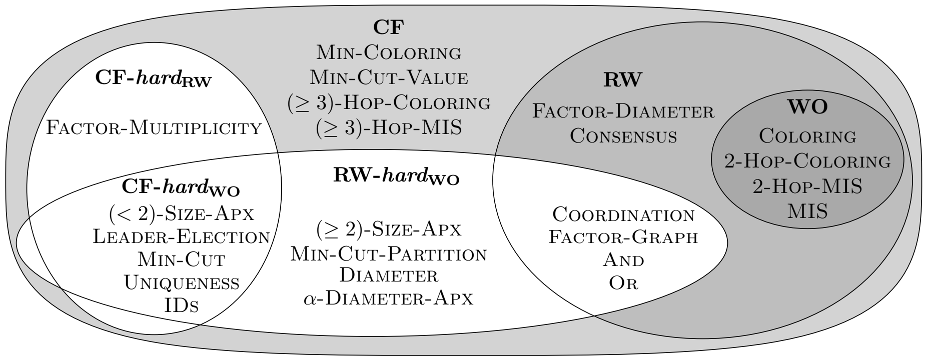

Fig. 1. Classes CF, RW and WO, and the respective hardness classes.

- And: nodes have input 0 or 1 and have to return the logical And of all inputs.
- Or: nodes have input 0 or 1 and have to return the logical Or of all inputs.
- Factor-Graph: agree on a mapping f inducing a factor F of the network graph. Each node v returns F and its corresponding node $f(v)$ in F.
- Factor-Diam: determine the diameter of a factor graph of the network.
- Factor-Multiplicity: determine the multiplicity of the smallest factor of the network.

The last three problems on this list require the notion of graph factors[5] which is introduced in the full version of this paper. Connections from distributed computability to graph factors were witnessed before, for example in [3]. For some problems on the list, namely k-Hop-MIS, k-Hop-Coloring, and α-Size-Apx, computability and/or hardness depends on the choice of k and α, respectively.

Of course for many problems on this list it is known whether they are contained in WO or CF \ WO. For example the well studied symmetry breaking tasks MIS or Coloring with $\Delta + 1$ colors (where Δ denotes the maximum degree of a node in the graph) are known to be in WO [29,2,28]. The work [17] presents WO-algorithms for each of the two problems 2-Hop-MIS and 2-Hop-Coloring. An example of a previously known hardness result is that an approximation α-Size-Apx with $\alpha < 2$ is sufficient to find unique identifiers with a WO-algorithm [34].

Example (Logical Or). Denote by ρ the output register of a node v. The following "algorithm" loosely solves Or. In the first round if node v has input 0, then it sets $\rho \leftarrow 0$, otherwise v sets $\rho \leftarrow \varepsilon$. In the second round all nodes set $\rho \leftarrow 1$ regardless of their input.

This method highlights how convenient Theorem 1 can be for an algorithm designer. The straight-forward solution however is no testimony to the crudeness of Or, since the following argument shows that it is indeed RW-*complete*$_{\text{WO}}$. We show how to turn a RW-algorithm $\mathcal{A}_{\text{RW}}$ solving $\Pi \in \text{RW}$ into a WO-algorithm $\mathcal{A}_{\text{WO}}$ that solves Π with access to an Or-oracle. In algorithm $\mathcal{A}_{\text{WO}}$ every node v will simulate one round of $\mathcal{A}_{\text{RW}}$ in every round; we denote v's simulated output register of $\mathcal{A}_{\text{RW}}$ by ρ_{RW}, and the actual output register of $\mathcal{A}_{\text{WO}}$ by ρ_{WO}. If in round r the register $\rho_{\text{RW}} = \varepsilon$, then v writes 1 to the input register of the oracle, otherwise it invokes the oracle with input 0. When the oracle answers 0 in round $r + 1$, the network was in a ready configuration in round r and v sets ρ_{WO} to the value contained in ρ_{RW} in round r.

5 Proof Sketch for Theorem 7

In this section we only present a sketch for the proof of Theorem 7; a comprehensive proof is presented in the appended full paper. Our proof is based on the techniques introduced in Section 2 and utilizes the aforementioned completeness result for Or. Theorem 7 states that if a problem Π is both CF-*hard*$_{\text{RW}}$ and

[5] In the distributed computing literature, the concept of graph factors was also referred to as covering graphs and graph lifts.

RW-$hard_{\mathrm{WO}}$, then it is also CF-$hard_{\mathrm{WO}}$. Let $\Pi \in$ CF-$hard_{\mathrm{RW}} \cap$ RW-$hard_{\mathrm{WO}}$ be a problem satisfying the premise. Denote by $\mathcal{A}_{\mathrm{LE}}$ a RW-algorithm solving Leader-Election with an access to a Π-oracle, and by $\mathcal{A}_{\mathrm{OR}}$ a WO-algorithm solving Or with an access to a Π-oracle respectively.

The idea is to design a WO-algorithm $\mathcal{A}$ solving Leader-Election with access to a Π-oracle by simulating one execution of $\mathcal{A}_{\mathrm{LE}}$ and multiple executions of $\mathcal{A}_{\mathrm{OR}}$, where the task of the latter is to determine whether the former has reached a ready configuration. That is, for every simulated round r of algorithm $\mathcal{A}_{\mathrm{LE}}$ a corresponding simulation $\mathcal{A}_{\mathrm{OR}}$, called the *fork* $[r]$ of $\mathcal{A}_{\mathrm{OR}}$, is initiated. The input to fork $[r]$ is 0 if v was ready in round r under $\mathcal{A}_{\mathrm{LE}}$ (v observes that from the simulated outcome of $\mathcal{A}_{\mathrm{LE}}$'s round r); the input is 1 otherwise. Since in $\mathcal{A}$ the Π-oracle can only be accessed once in every round, algorithm $\mathcal{A}$ uses a careful mechanism to schedule disjoint accesses by the simulated execution of $\mathcal{A}_{\mathrm{LE}}$ and all forks to this scarce resource; we refer to the full version for the details.

The logic of Or guarantees that fork $[r]$ of $\mathcal{A}_{\mathrm{OR}}$ has output 0 if and only if round r under $\mathcal{A}_{\mathrm{LE}}$'s simulation is in a ready configuration. Since $\mathcal{A}_{\mathrm{OR}}$ is a WO-algorithm, node v can immediately rely on a returned 0 value to conclude that this indeed happened. Employing Lemma 4, one can assume that $\mathcal{A}_{\mathrm{LE}}$ sustainably solves the leader election problem, thus ensuring that the output returned in v's simulated round r of $\mathcal{A}_{\mathrm{LE}}$ yields a correct output for Leader-Election. This establishes Theorem 7.

References

1. Abrahamson, K., Adler, A., Higham, L., Kirkpatrick, D.: Probabilistic solitude verification on a ring. In: PODC (1986)
2. Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. Journal of Algorithms 7(4), 567 – 583 (1986)
3. Angluin, D.: Local and global properties in networks of processors (extended abstract). In: Theory of computing (1980)
4. Angluin, D., Aspnes, J., Chan, M., Fischer, M., Jiang, H., Peralta, R.: Stably computable properties of network graphs. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS (2005)
5. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: PODC (2004)
6. Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: PODC (2006)
7. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing 20, 279–304 (2007)
8. Angluin, D., Fischer, M., Jiang, H.: Stabilizing consensus in mobile networks. In: Gibbons, P., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS (2006)
9. Aspnes, J., Ruppert, E.: An introduction to population protocols. In: Garbinato, B., Miranda, H., Rodrigues, L. (eds.) MiNEMA (2009)
10. Boldi, P., Vigna, S.: Computing anonymously with arbitrary knowledge. In: PODC (1999)

11. Boldi, P., Vigna, S.: An effective characterization of computability in anonymous networks. In: DISC (2001)
12. Boldi, P., Vigna, S.: Universal dynamic synchronous self–stabilization. Distributed Computing 15(3), 137–153 (2002)
13. Chalopin, J., Das, S., Santoro, N.: Groupings and pairings in anonymous networks. In: DISC (2006)
14. Chalopin, J., Godard, E., Métivier, Y.: Local terminations and distributed computability in anonymous networks. In: Taubenfeld, G. (ed.) Distributed Computing (2008)
15. Dolev, S.: Self-Stabilization (2000)
16. Dolev, S., Israeli, A., Moran, S.: Uniform dynamic self-stabilizing leader election. In: Toueg, S., Spirakis, P., Kirousis, L. (eds.) Distributed Algorithms (1992)
17. Emek, Y., Wattenhofer, R.: Stone age distributed computing. In: PODC (2013)
18. Flocchini, P., Kranakis, E., Krizanc, D., Luccio, F.L., Santoro, N.: Sorting and election in anonymous asynchronous rings. J. Parallel Distrib. Comput. 64(2), 254–265 (Feb 2004)
19. Fraigniaud, P., Korman, A., Peleg, D.: Local distributed decision. In: FOCS (oct 2011)
20. Fraigniaud, P., Halldórsson, M., Korman, A.: On the impact of identifiers on local decision. In: Baldoni, R., Flocchini, P., Binoy, R. (eds.) OPODIS (2012)
21. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Oracle size: A new measure of difficulty for communication tasks. In: PODC (2006)
22. Fraigniaud, P., Korman, A., Parter, M., Peleg, D.: Randomized distributed decision. In: DISC (2012)
23. Göös, M., Suomela, J.: Locally checkable proofs. In: PODC (2011)
24. Guerraoui, R., Ruppert, E.: What can be implemented anonymously? In: DISC (2005)
25. Itai, A., Rodeh, M.: Symmetry breaking in distributive networks. In: FOCS (1981)
26. Korman, A., Kutten, S., Peleg, D.: Proof labeling schemes. In: PODC (2005)
27. Korman, A., Sereni, J.S., Viennot, L.: Toward more localized local algorithms: removing assumptions concerning global knowledge. In: PODC (2011)
28. Linial, N.: Locality in distributed graph algorithms. SIAM Journal on Computing 21(1), 193–201 (1992)
29. Luby, M.: A simple parallel algorithm for the maximal independent set problem. In: Theory of computing (1985)
30. Lynch, N.A.: Distributed Algorithms (1996)
31. Mavronicolas, M., Michael, L., Spirakis, P.: Computing on a partially eponymous ring. In: OPODIS (2006)
32. Métivier, Y., Robson, J.M., Zemmari, A.: Analysis of fully distributed splitting and naming probabilistic procedures and applications. In: Moscibroda, T., Rescigno, A. (eds.) SIROCCO (2013)
33. Naor, M., Stockmeyer, L.: What can be computed locally? SIAM Journal on Computing 24(6), 1259–1277 (1995)
34. Schieber, B., Snir, M.: Calling names on nameless networks. In: PODC (1989)
35. Yamashita, M., Kameda, T.: Computing on anonymous networks: Part i-characterizing the solvable cases. IEEE Trans. Parallel Distrib. Syst. 7(1), 69–89 (Jan 1996)

Computability in Anonymous Networks: Revocable vs. Irrecovable Outputs [Full Version]

Yuval Emek[1], Jochen Seidel[2], and Roger Wattenhofer[2]

[1]Faculty of Industrial Engineering and Management, Technion, Israel
`yemek@ie.technion.ac.il`

[2]Distributed Computing, ETH Zurich, Switzerland
`{seidelj, wattenhofer}@ethz.ch`

Abstract

What can be computed in an anonymous network, where nodes are not equipped with unique identifiers? It turns out that the answer to this question depends on the commitment of the nodes to their first computed output value: Two classes of problems solvable in anonymous networks are defined, where in the first class nodes are allowed to revoke their outputs and in the second class they are not. These two classes are then related to the class of all centrally solvable network problems, observing that the three classes form a strict linear hierarchy, and for several classic and/or characteristic problems in distributed computing, we determine the exact class to which they belong.

Does this hierarchy exhibit complete problems? We answer this question in the affirmative by introducing the concept of a distributed oracle, thus establishing a more fine grained classification for distributed computability which we apply to the classic/characteristic problems. Among our findings is the observation that the three classes are characterized by the three pillars of distributed computing, namely, local symmetry breaking, coordination, and leader election.

1 Introduction

We study computability in networks, referred to hereafter as *distributed computability.* Distributed computability is equivalent to classic centralized (Turing Machine) computability when the nodes are equipped with unique (comparable) identifiers. However, as Angluin noticed in her seminal work [3], distributed computability becomes fascinating in *anonymous* networks, where nodes do not have unique IDs. What can be computed with deterministic algorithms merely depends on the topology of the network, and it is well known that problems like maximal independent set can be solved in an anonymous network only if the nodes are allowed to toss coins. We therefore consider the distributed computability of *randomized* algorithms running in anonymous networks. Notice that in the scope of this paper, we do not impose any limitations on the complexity resources (time, message/memory size, ...), however, like in classic sequential computability theory, we do require a correct result after a finite amount of time.

Apart from its theoretical interest, the study of anonymous networks is motivated by various real-world scenarios. For example, the nodes may be indistinguishable due to their fabrication in a large-scale industrial process [5], in which equipping every node with a unique identifier (serial number) is not economically feasible. In other cases nodes may not wish to reveal their unique identity out of privacy and security concerns [26].

1.1 Setting

Distributed Problems. We consider simple (undirected, loop-free and no parallel edges) connected finite graphs G, and denote the node and edge sets of a graph G by $V(G)$ and $E(G)$ or V and E if G is clear from the context. A function $f : V(G) \to L$ is called a *labeling* of the graph G, and we refer to the set L as the set of *values* that f assigns to nodes in G. A *distributed problem* Π is a set of three-tuples (G, i, o), where G is a graph as described above, and i and o are *input labels* and *output labels* for G. For every problem there are two sets $I(\Pi)$ and $O(\Pi)$ denoting the *input values* and *output values* of Π, i.e., the values that the labels i and o assign, correspondingly. Such a three-tuple $(G, i, o) \in \Pi$ is called a *(solved) instance* of Π. An *input instance* of Π is a two-tuple (G, i) for which there exists a *valid output* o satisfying $(G, i, o) \in \Pi$, and we also write $(G, i) \in \Pi$ for input instances of the problem. We restrict ourselves to problems that are solvable in a centralized setting.

Randomized Anonymous Algorithms. Our definition of how distributed algorithms work follows the convention of [36] for synchronized network systems (message passing) with simultaneous starting times. Nodes execute the same randomized and uniform algorithm in synchronous *rounds*, and in each round we allow each node access to finitely many random bits. Every node v knows its degree $\deg(v)$ and can distinguish between its neighbors $\Gamma(v)$ (by means of a bijection $\{1, \ldots, \deg(v)\} \to \Gamma(v)$, cf. the *port model*). In each round every node sends and receives a message of unbounded, yet finite, size to and from each individual neighbor. To ease our discussion every node v is equipped with one *input register* holding some problem-dependent input value and one *output register*. The output register initially contains a special symbol ε indicating v *is not ready* to return an output. Any value $x \neq \varepsilon$ contained in v's output register is interpreted as v being *ready* to return its output and we say that v *has output* x. A global *configuration* in which all nodes are ready is called a *ready configuration*. When algorithm $\mathcal{A}$ is in a ready configuration, we define *$\mathcal{A}$'s output* $o_{\mathcal{A}} : V(G) \to O$ by setting $o_{\mathcal{A}}(v)$ to be the content of node v's output register. In the following, we consider two different notions of *output revocability*.

Definition (Output Revocability)**.** An algorithm is referred to as a *write-once* algorithm if every node is restricted to write to its output register at most once. If this restriction is lifted, then we call it a *rewrite* algorithm.

In other words, in a rewrite algorithm a node may revoke its output, e.g., by writing ε to its output register. While every execution of a write-once algorithm reaches at most one ready configuration, during the execution of a rewrite algorithm many ready configurations can occur. Note that the converse does not hold: an algorithm that is guaranteed to reach at most one ready configuration is not necessarily a write-once algorithm. In the existing literature, algorithms are typically considered to be write-once algorithms.

Definition (Correctness)**.** Fix some problem Π and an algorithm $\mathcal{A}$. A ready configuration of $\mathcal{A}$ when invoked on an input instance $(G, i) \in \Pi$ is said to be *valid* if the output $o_{\mathcal{A}}$ of $\mathcal{A}$ in this configuration is a valid output for (G, i). Algorithm $\mathcal{A}$ is said to *solve* Π if it satisfies the following two conditions for every input instance $(G, i) \in \Pi$:

1. A ready configuration is reached within finite time with probability 1.
2. Every ready configuration that can occur with a positive probability is valid.

The aforementioned definition of correctness requires that all occurring ready configurations will be correct (i.e., correspond to a valid output). In Section 2 we show that our definition of correctness is robust to certain changes. Notice that in the scope of this paper, we do not require that an algorithm *terminates* in order to be correct. However, the algorithms designed throughout the paper do terminate, and the general transformation techniques we present (i.e., compilers/simulations) can be designed to ensure termination if the algorithms to which the transformation is applied terminate.

The choice of output revocability has a significant impact on the problems that an algorithm can solve. In the following the terms WO-algorithms and RW-algorithms will thus be used to denominate write-once and rewrite algorithms running in an anonymous network, respectively; RW and WO refer to the *classes* of distributed problems solvable by these two types of algorithms. Lastly, we denote by CF the class of distributed problems that are solvable in a centralized setting (by a Turing machine), bearing in mind that this class essentially includes every *computable function* on graphs. The distinction of these classes is justified by the following observation.

Observation. *The classes of distributed problems satisfy* WO $\subset$ RW $\subset$ CF *(in the strict sense).*

Proof. RW $\subset$ CF: A Turing machine can simulate an algorithm running in an anonymous network. On the other hand, the techniques from [3] can be used to show that leader election is not in RW, but it is clearly possible in the centralized setting.

WO $\subset$ RW: It is clear from the definition that every WO-algorithm is also a RW-algorithm. In Section 4.1.2, we will show that for example Consensus is in RW, but not in WO.

□

1.2 Our Contribution

What can be computed in anonymous networks? As it turns out the effect output revocability has on the distributed computability of anonymous networks is remarkable. A total of 21 problems, including some of the most fundamental problems in distributed computing, are classified according to the exact class to which they belong (Section 4).

Does the hierarchy we present exhibit complete problems? To answer this question we introduce the notion of accessing an oracle in a distributed setting and show that this notion is sound (Section 3). As the first stepping stone in this effort we show that the classes WO and RW are robust against two modifications to the aforementioned correctness condition (Section 2). Each of our 21 problems is then classified according to its *hardness* or *completeness* for the three classes (Section 4.2), thus obtaining a deeper understanding of the intrinsic properties of these problems. Surprisingly, the WO, RW, and CF classes turn out to capture exactly the three *pillars* of distributed computing, namely, local symmetry breaking, coordination, and leader election, respectively.

1.3 Related Work

The history of distributed computability starts with the work of Angluin [3] proving that randomization does not help to elect a leader in anonymous networks. Later, it was shown that electing a leader in an anonymous ring network is possible if the size n of the ring is known [30], in fact, a $(2-\varepsilon)$-approximation of n is enough [1], not only in the special case of a ring but in general networks [40]. It turns out that all these results (and many similar ones) come almost for free once our graph-theoretic characterization for the class RW is established. The connection between computation in anonymous networks and products of graphs (graph coverings) which was first observed in Angluin's seminal work plays an important role in this characterization.

There is a line of work that concentrates on *deterministic* distributed algorithms for problems in CF, in particular if some parameters of the topology of the graph (for instance, its size) are known, e.g. [42, 10]. Deterministic algorithms are interesting to investigate even if the graph is restricted to a ring [18, 13], and also assignments of not necessarily unique identifiers were studied in this context [37].

Another line of research studies computability in anonymous (directed) networks in connection with termination. Not unlike us it is argued that termination in distributed systems is an issue that is not directly evident, since one may be interested in systems where nodes terminate independently of others. Different forms of termination and prior knowledge are studied in this line of work, where the strongest anonymous model considered is equivalent to deterministic write-once algorithms with knowledge of an upper bound to the network size [11]. When no prior knowledge is assumed the class of solvable problems can be fully characterized using *local views*[1] (quasi-coverings) and recursive functions [14]. Extending their approach,

[1]See Section 4.1.1 for a definition.

in the context of the current paper an individual node executing a RW-algorithm can never be entirely sure about termination. We show that the class RW lies *between* the two classes WO (local termination) and CF (global termination).

Output revocability should not be confused with the concept of *eventual correctness*, where the network eventually converges to a correct output. For example, *self-stabilizing algorithms* [15] allow the system to return an incorrect output for a finite amount of time, thus allowing a fault-tolerant algorithm to recover from errors. With randomization, self-stabilizing leader election is possible on general graphs [16], hence with randomization every CF-problem is eventually solvable in an anonymous network. In our terminology eventual correctness could be viewed as requiring that some ready configuration, not necessarily the first one, is stable[2] and valid. We require though that an output is returned after finite time and that every output returned by the network is correct, but we do allow the network to revoke partial outputs. The problems solvable by self-stabilizing algorithms in directed graphs can be characterized by *fibrations* [12], the directed analog to factors[1] of graphs.

The self-stabilization concept is also used in the scope of population protocols, introduced by Angluin et al. [5]. Population protocols are an example for asynchronous distributed automata with restricted computational power. In this model, nodes cannot do arbitrary computations, as they are modeled by finite state machines, see [9] for an overview. Regarding computability in a clique network, [5, 6, 7] conclude the predicates solvable to be exactly those expressible in first-order Presburger arithmetic. On graphs with bounded degree a Turing machine with linearly bounded space can be simulated [4]. It was also studied how the correctness condition for population protocols affects solvability of the CONSENSUS problem [8].

Apart from these results, not much is known about distributed computability, as a large fraction of research deals with complexity rather than computability. However, there are surprising connections between complexity and computability, which go beyond us borrowing the terms hardness and completeness. Regarding network algorithms, in the last thirty years, a lot of research went into the question how fast a particular problem can be computed by the network. Literally hundreds of new upper and lower bounds have been found. The fastest algorithms deliver a result within constant time, independent of the size of the network, see [41] for a recent survey. It is intriguing that our research which is about computability has most connections to this "fastest" class of distributed algorithms.

Naor and Stockmeyer [39] introduced the notion of locally checkable labelings (essentially an apply-once oracle) in identified networks and ask the question how a constant-time deterministic algorithm can decide whether the labeling represents a correct solution to a given problem. Follow-up work looked at the bit complexity required to solve decision problems [31] and a problem hierarchy depending on the size of checkable labelings was suggested [25], also for anonymous networks. Our work also yields a characterization of decision problems in RW. How apply-once oracles can be used to make broadcast and wake-up schemes more efficient was studied in [21]. However, we do not restrict the run-time to be constant and allow randomization for symmetry breaking. Pruning algorithms [32] that build a solution gradually in a write-once fashion were inspired by the same line of research, in an effort to remove the necessity of global knowledge about the graph. While our algorithms are required to give a correct output in every execution, [19, 22] study the notion of (p, q)-decidable decision problems (an anonymous randomized algorithm is allowed to return a wrong output with constant probability) and find a strict hierarchy among the classes of solvable problems depending on the success probabilities. If a randomized algorithm is allowed to fail (Monte-Carlo algorithm), then a leader can be elected [38] with high probability (w.h.p, i.e., with probability $1 - n^{-c}$ for any c). Hence any CF-problem can be solved in an anonymous network with high probability, whereas we require a correct output with probability 1.

Non-deterministic algorithms running in an anonymous setting can fully determine the structure of the radius t-ball around itself in [20], and thus solve exactly the decision problems that are closed under so-called *t-homomorphisms*, that is, homomorphisms that preserve the structure t hops around every node, regardless of access to unique identifiers. In our model only the local view can be retrieved. It may thus be surprising that RW-algorithms can solve exactly the problems that such non-deterministic constant-time algorithms can solve in a single round.

[2]A configuration is said to be *stable* if the nodes no longer revoke their outputs, see Section 2.

Lastly, it is worth mentioning that in the context of *shared memory* systems a notion of distributed oracles in an asynchronous environment is studied. Usually such an oracle is applied once to implement a protocol (algorithm), and the tasks (e.g., consensus) also form hierarchies by their ability to implement each other [27, 34, 23]. Unlike in our model, computability in shared memory systems is hindered by asynchronous execution rather than the network structure and has surprising connections to topology [28]. Nonetheless variants of the consensus tasks turn out to be complete for the class RW.

2 Notions of Correctness

Our definition of a correct algorithm requires every ready configuration that occurs throughout an execution to be valid. For WO-algorithms this requirement is superfluous since its execution will reach at most one ready configuration. However, RW-algorithms may invalidate or change a ready configuration after it occurred. One may therefore wonder if strengthening the definition by allowing only one durable ready configuration makes the class of solvable problems strictly smaller. On the other hand one may be tempted to weaken this definition, in hope to capture a larger class of problems by requiring only the first occurring ready configuration to be correct. Perhaps surprisingly we show that these two variants have no effect and are equivalent to the current definition of correctness. This equivalence will play a key role when we reason about RW-algorithms in the next section which covers distributed oracles.

Definition (Sustainable Correctness)**.** A ready configuration is said to be *stable*, if the nodes no longer revoke their outputs. Algorithm $\mathcal{A}$ is said to *sustainably solve* a problem Π if it satisfies the following two conditions for every input instance $(G, i) \in \Pi$:
1. A ready configuration is reached within finite time with probability 1.
2. The first ready configuration that occurs is valid and stable.

Definition (Loose Correctness)**.** Algorithm $\mathcal{A}$ is said to *loosely solve* a problem Π if it satisfies the following two conditions for every input instance $(G, i) \in \Pi$:
1. A ready configuration is reached within finite time with probability 1.
2. The first ready configuration that occurs is valid.

The class *Sustainable-RW* (respectively, *Loose-RW*) consists of every distributed problem that can be sustainably solved (resp., loosely solved) by a RW-algorithm. Since sustainable correctness (resp., loose correctness) is a restriction (resp., a relaxation) of correctness as defined in Section 1.1, we conclude that Sustainable-RW $\subseteq$ RW $\subseteq$ Loose-RW. Note that the corresponding classes *Sustainable-WO* and *Loose-WO* for WO-algorithms are equal to the class WO due to the write-once restriction of these algorithms. The following theorem states that also for RW-algorithms the three classes are, in fact, equal.

Theorem 1. *The classes of problems solvable by* RW*-algorithms under the three different notions of correctness satisfy Sustainable-*RW = RW = *Loose-*RW*.*

The proof of Theorem 1 is based on a simple concept referred to as *safe broadcast* in which information is broadcast throughout the whole network and no ready configuration is reached before all nodes have received the information. When a node v receives a previously unseen message M that should be safely broadcast, it writes ε to its output register for at least one round and forwards M to all its neighbors. This ensures that M propagates through the network together with a front of non-ready nodes, so that no ready configuration can be reached during the dissemination of M.

Based on the safe broadcast concept, we develop a generic technique called *inhibiting messages* which will also be useful when designing algorithms in Section 4. For every node v, this programming technique employs a register ρ, usually chosen to be v's output register, and a list L containing pairs (i, x) where i is an integer, typically a round or phase number, and x is an arbitrary value. Two methods are provided for every node v, where the invocation of these methods is determined by user defined conditions: A node v can (1) append a new pair (i, x) to L; and (2) broadcast an inhibiting message M_i for i. The operation is as follows. If v sends or receives an inhibiting message M_i, then for all x the pairs (i, x) are removed from

L. Whenever L is empty, node v sets $\rho \leftarrow \varepsilon$. Assuming that L is non-empty, denote by $(i_{\min}, x_{\min})$ a pair in Q that satisfies $i_{\min} \leq i$ for all pairs (i, x) in Q. In that case, the default value stored in ρ is the value $x_{\min}$. The one exception to this rule occurs when v receives an inhibiting message $M_{i_{\min}}$, in which case v sets $\rho \leftarrow \varepsilon$ in the current round, which means that ε is written to ρ between any two consecutive non-ε values. Notice that the front of non-ready nodes propagates through the network with the inhibiting message M_i only as long as M_i *invalidates* the output currently contained in the output registers.

We employ inhibiting messages to show that the class RW is robust against the stated modifications to the definition of a correct algorithm. The proof of Theorem 1 relies on a *sustainability compiler* that takes a RW-algorithm $\mathcal{A}$ that loosely solves problem Π and transforms it into a RW-algorithm $\hat{\mathcal{A}}$ that sustainably solves this problem. Specifically, under algorithm $\hat{\mathcal{A}}$, every node v simulates $\mathcal{A}$; to avoid confusion, let $\hat{\rho}$ be v's output register under $\hat{\mathcal{A}}$ and let ρ be v's register simulating the output register of $\mathcal{A}$. The compiler is based on sending inhibiting messages, where the register upon which the inhibiting message technique operates is $\hat{\rho}$ and the integers i of the technique are identified with the round numbers. In every round r, if v is not ready in round r under $\mathcal{A}$, then node v broadcasts an inhibiting message M_r, that is, v broadcasts an inhibiting message for r if $\rho = \varepsilon$. If on the other hand v's register ρ contains the value $x \neq \varepsilon$ in round r, then v appends the pair (r, x). Theorem 1 is established by proving the following lemma.

Lemma 2. *Let $\mathcal{A}$ be a* RW*-algorithm loosely solving a problem* Π *and let $\hat{\mathcal{A}}$ be the* RW*-algorithm obtained by applying the sustainability compiler to $\mathcal{A}$. Then $\hat{\mathcal{A}}$ sustainably solves* Π.

Proof. Consider some input instance $(G, i) \in \Pi$ and denote by η the execution of $\mathcal{A}$ on (G, i) that $\hat{\mathcal{A}}$ simulates. Algorithm $\hat{\mathcal{A}}$ employs inhibiting messages. For the sake of the analysis let $i_v(r)$ denote the value $i_{\min}$ of node v in round r, or NIL if v's queue is empty. In particular, if $i_v(r) \neq \text{NIL}$, then the value stored in v's output register $\hat{\rho}$ is the output of v in round r of η. By definition, η must reach a ready configuration and the first ready configuration reached by η is valid; let r_0 denote the round in which this valid ready configuration is reached and let o_0 be the valid output returned by η in that round. Notice that under algorithm $\hat{\mathcal{A}}$, no node broadcasts an inhibiting message for round r_0, whereas at least one node broadcasts an inhibiting message M_r for every round $r < r_0$. This implies that under $\hat{\mathcal{A}}$, eventually $i_v(r) = r_0$ for every node v; let $r_1 \geq r_0$ be the first round in which this happens. Starting from round r_1, algorithm $\hat{\mathcal{A}}$ outputs o_0 and the design of the inhibiting message technique guarantees that $\hat{\mathcal{A}}$ will not revoke this output. Therefore, we only have to ensure that under $\hat{\mathcal{A}}$, in all rounds $r < r_1$ at least one node is not ready.

To that end, assume for the sake of contradiction that there exists a round $r < r_1$ in which all nodes are ready under $\hat{\mathcal{A}}$. In that case $i_v(r) \neq \text{NIL}$ for every node v. If $i_v(r) = i_u(r)$ for all $u, v \in V(G)$ then $\mathcal{A}$ was in a ready configuration in round r and thus $r = r_0 = r_1$. Therefore in round r under $\hat{\mathcal{A}}$, there must be nodes having outputs from two different rounds of η. Moreover, since G is connected there must exist two such nodes u and v, $\{u, v\} \in E(G)$. Since the sustainability compiler employs the inhibiting message technique, we conclude that $i_u(r) \neq i_v(r)$ and without loss of generality assume that $i_u(r) < i_v(r)$. But this means, that in some round $r' < r$ node v sent an inhibiting message for round $i_u(r)$ and this message reaches u in round $r' + 1 \leq r$, in contradiction to the assumption that round $i_u(r)$ is non-inhibited for u in round r. It follows that $\hat{\mathcal{A}}$ does not reach a ready configuration prior to round r_1 which completes the proof. □

3 Distributed Oracles

In this section, we introduce the concepts of hardness and completeness, which are central to this work and allow us to gain a deeper understanding how the computability classes relate to each other. To that end, we introduce the notion of an oracle working in a distributed setting.

Definition (Algorithm with access to a Π-oracle)**.** Consider some problem Π. A C-algorithm, $\text{C} \in \{\text{WO}, \text{RW}\}$, with *access to a Π-oracle* is a distributed C-algorithm in which every node v is equipped with a designated *oracle input register* and a designated *oracle output register*. Given some $r \geq 1$, let $\tilde{i}(v)$ be the content of v's oracle input register in round r and let $\tilde{o}(v)$ be the content of v's oracle output register in round $r + 1$. If $(G, \tilde{i})$ is an input instance of Π, then it is guaranteed that $\tilde{o}$ is a valid output for $(G, \tilde{i})$. No assumptions are made on the operation of the algorithm if $(G, \tilde{i}) \notin \Pi$.

While applying the oracle in every round of the algorithm may seem powerful, allowing the distributed algorithm to arbitrarily choose the rounds in which the oracle is applied may require some sort of global coordination, which is not necessarily possible. In comparison, a weaker definition of "accessing an oracle" would be to allow application of the oracle only once in round 1. This distinction does not make a difference for problems Π without inputs ($|I(\Pi)| = 1$), e.g., for graph theoretic problems like coloring, maximal independent set, or determining the diameter, because the oracle is always applied on the same input instance. It does however affect problems that do receive inputs ($|I(\Pi)| \geq 2$), e.g., CONSENSUS or logical AND and OR. It will be convenient to refer to this weaker manner of accessing an oracle as accessing an *apply-once* oracle.

As stated above, based on the oracle concept, we will soon introduce the notion of hard and complete problems for the hierarchy of problem classes. This notion would be ill-defined if accessing an oracle to a problem $\Pi_{\mathrm{C}} \in \mathrm{C}$ could enhance the computational power of a C-algorithm. We ensure that the notion of an algorithm with access to an oracle is sound in the following theorem. Note that the statement of the theorem does not mention the case $\mathrm{C} = \mathrm{CF}$, since the soundness of oracles for centralized models is well understood and in any case, beyond the scope of the current paper.

Theorem 3 (Soundness)**.** *If a problem Π is solvable by a C-algorithm, $\mathrm{C} \in \{\mathrm{RW}, \mathrm{WO}\}$, accessing an oracle to a problem $\Pi_{\mathrm{C}} \in \mathrm{C}$, then Π can also be solved by a C-algorithm that does not access any oracle.*

The key to proving this theorem is to show that in a C-algorithm $\mathcal{A}^{\mathrm{a}}$ that solves a problem Π with *access* to a Π_{C}-oracle, $\Pi_{\mathrm{C}} \in \mathrm{C}$, one can *replace* the oracle access by simulating a C-algorithm $\mathcal{A}^{\mathrm{r}}$ that solves Π_{C} without any oracle access. We will first prove that accessing apply-once oracles does not enhance the computational power of RW- and WO-algorithms, since the two algorithms $\mathcal{A}^{\mathrm{a}}$ and $\mathcal{A}^{\mathrm{r}}$ can be executed consecutively one after the other, or in other words, that algorithm $\mathcal{A}^{\mathrm{a}}$ accessing an apply-once oracle can be simulated without accessing an oracle by executing $\mathcal{A}^{\mathrm{r}}$ first. This turns out to be a non-trivial task especially for RW-algorithms since a node v simulating $\mathcal{A}^{\mathrm{r}}$ cannot know for sure that the output returned by $\mathcal{A}^{\mathrm{r}}$ will not be revoked later on, i.e., whether it can be safely used for the execution of $\mathcal{A}^{\mathrm{a}}$. It therefore does not know when such a result is valid so that a simulation of $\mathcal{A}^{\mathrm{a}}$ can be invoked based on this result. The technique we present to resolve this issue for RW-algorithms is based on Theorem 1. Actually, we will need an extension of Lemma 2 (the key to the proof of Theorem 1) to RW-algorithms accessing a Π'-oracle for some problem Π'. To that end, we observe that the construction of the sustainability compiler and the arguments used in the proof of Lemma 2 can be repeated with no changes to yield the following.

Lemma 4. *Fix some problem Π'. Let $\mathcal{A}$ be a RW-algorithm with access to a Π'-oracle loosely solving a problem Π and let $\hat{\mathcal{A}}$ be the RW-algorithm with access to a Π'-oracle obtained by applying the sustainability compiler to $\mathcal{A}$. Then $\hat{\mathcal{A}}$ sustainably solves Π with an access to a Π'-oracle.*

In other words, Lemma 4 states that the three notions of correctness for RW-algorithms are equivalent even when the algorithm has an access to a Π'-oracle for some (arbitrary) problem Π'. This enables us to establish the following lemma that states the soundness of apply-once oracles for RW-algorithms.

Lemma 5 (Consecutive RW Execution)**.** *Let Π_{C} be a problem in RW and let $\mathcal{A}^{a}$ be a RW-algorithm solving an arbitrary problem Π with access to an apply-once Π_{C}-oracle. Then Π is solvable by a RW-algorithm without access to any oracle.*

Proof. Let $\mathcal{A}^{\mathrm{r}}$ be a RW-algorithm solving Π_{C}. Employing Lemmas 2 and 4, we assume that $\mathcal{A}^{\mathrm{r}}$ and $\mathcal{A}^{\mathrm{a}}$ sustainably solve Π_{C} and Π, respectively. We would like to show that $\Pi \in \mathrm{RW}$ by designing a RW-algorithm $\mathcal{A}$ that solves Π without access to any oracle. This will be accomplished by letting $\mathcal{A}$ simulate the execution of $\mathcal{A}^{\mathrm{a}}$, using $\mathcal{A}^{\mathrm{r}}$ to replace $\mathcal{A}^{\mathrm{a}}$'s access to the apply-once Π_{C}-oracle. Algorithm $\mathcal{A}$ faces the issue that an output returned to a node v by $\mathcal{A}^{\mathrm{r}}$ may not be part of a ready configuration and thus it is not clear whether v should use this value as an output of the Π_{C}-oracle that $\mathcal{A}^{\mathrm{a}}$ invoked. To cope with that, algorithm $\mathcal{A}$ performs a systematic search for some round in which $\mathcal{A}^{\mathrm{r}}$ reaches a ready configuration.

Algorithm $\mathcal{A}$ simulates algorithms $\mathcal{A}^{\mathrm{r}}$ and $\mathcal{A}^{\mathrm{a}}$; to avoid confusion, let ρ, ρ^{r}, and ρ^{a} denote the output registers of v under $\mathcal{A}, \mathcal{A}_1$, and $\mathcal{A}^{\mathrm{a}}$, respectively. Algorithm $\mathcal{A}$ works in phases, where phase p consists of $2p$

rounds as follows. In each phase p, every node v first simulates p rounds of $\mathcal{A}^{\mathrm{r}}$; the role of this simulation is to replace the access to the (apply-once) Π_{C}-oracle. While this simulation takes place, node v sets $\rho \leftarrow \varepsilon$ ensuring that a ready configuration can only be reached in the second half of phase p. Node v is referred to as *sad* if $\rho^{\mathrm{r}} = \varepsilon$ at the end of round p of phase p; otherwise, node v is referred to as *happy*. If node v is sad, then it does not participate in the next p rounds of phase p and sets its output register $\rho \leftarrow \varepsilon$ in round $2p + 1$. If node v is happy, then in the next p rounds of phase p, it simulates p rounds of $\mathcal{A}^{\mathrm{a}}$ using the value stored in ρ^{r} as the output of the Π_{C}-oracle (accessed by $\mathcal{A}^{\mathrm{a}}$) and sets $\rho = \rho^{\mathrm{a}}$ in every round of the simulation. For convenience, let σ^{a}_p denote the sequence of rounds (of $\mathcal{A}$'s execution) that are dedicated to simulating algorithm $\mathcal{A}^{\mathrm{a}}$ in phase p, i.e., $\sigma^{\mathrm{a}}_1 = [2], \sigma^{\mathrm{a}}_2 = [5, 6]$ and so on. It will be important for the analysis that when simulating algorithms $\mathcal{A}^{\mathrm{r}}$ and $\mathcal{A}^{\mathrm{a}}$ in phase $p + 1$, node v reuses the same random bits that were used in phase p to which v only adds the random bits required for the simulation of round $p + 1$ in both algorithms.

For the sake of the analysis, let η^{r} be the execution of algorithm $\mathcal{A}^{\mathrm{r}}$ that corresponds to the simulation performed by algorithm $\mathcal{A}$. Notice that η^{r} is well-defined since under $\mathcal{A}$, the simulation of $\mathcal{A}^{\mathrm{r}}$ reuses the same random bits in every phase, so that in all phases p, the first p rounds of $\mathcal{A}^{\mathrm{r}}$ correspond to the first p rounds of η^{r}. Denote by o^{r} the output obtained from the stable ready configuration reached by η^{r}. Based on that, let η^{a} be the execution of algorithm $\mathcal{A}^{\mathrm{a}}$ that corresponds to the simulation performed by algorithm $\mathcal{A}$ in which the oracle access is replaced by o^{r}, and let o^{a} denote the output obtained from the stable ready configuration reached by η^{a}. The execution η^{a} and its output o^{a} are well defined since $\mathcal{A}^{\mathrm{a}}$ sustainably solves Π and $\mathcal{A}$ reuses random bits to simulate $\mathcal{A}^{\mathrm{a}}$ as well. Denote by t^{r} and t^{a} the rounds in which the stable ready configurations of η^{r} and η^{a} are reached for the first time, respectively, and let $t = \max\{t^{\mathrm{r}}, t^{\mathrm{a}}\}$. We argue that algorithm $\mathcal{A}$ reaches the first ready configuration in phase t, namely in round $\sigma^{\mathrm{a}}_t(t^{\mathrm{r}})$ of $\mathcal{A}$'s execution, and that the output of this ready configuration is o^{a}, which together with Theorem 1 establishes the assertion since $\mathcal{A}$ (at least) loosely solves Π.

To see that this is indeed true recall that under algorithm $\mathcal{A}$, a node v may only set ρ to a non-ε value in the second half of a phase that is dedicated to simulating $\mathcal{A}^{\mathrm{a}}$. In phases $p < t^{\mathrm{r}}$ at least one node is sad, i.e., not ready in round p of η^{r}, and therefore not ready during the second half of phase p. On the other hand, in phases $p \geq t^{\mathrm{r}}$ all nodes are happy and the simulation of $\mathcal{A}^{\mathrm{a}}$ performed by $\mathcal{A}$ corresponds to the first p rounds of η^{a}. The correctness of $\mathcal{A}$ now follows from the sustainable correctness of $\mathcal{A}^{\mathrm{a}}$. □

The crux in the proof of Lemma 5 was to show how two RW-algorithms can be executed consecutively in a correct manner. At first, it seems that the same technique is inapplicable to a WO-algorithm (accessing an apply-once oracle), since under algorithm $\mathcal{A}$ described in the proof of Lemma 5, a node will revoke any output it returned in the last round of a phase, i.e., algorithm $\mathcal{A}$ is not a WO-algorithm due to our construction. However the technique can be slightly modified so that it is applicable to WO-algorithms as well.

To address the aforementioned issue, we make three adjustments to the construction of algorithm $\mathcal{A}$ when it is applied to a WO-algorithm $\mathcal{A}^{\mathrm{a}}$ with access to a Π_{C}-oracle, $\Pi_{\mathrm{C}} \in \mathrm{WO}$. To describe the adjustments we use the same terminology as in the proof of Lemma 5: (1) Node v is not allowed to change the value stored in its output register ρ after the first value $x \neq \varepsilon$ was written to it. (2) If v is sad at the end of round p of phase p, then v broadcasts a *sadness message* for phase p. (3) If a happy node v in phase p receives a sadness message for that phase (in one of the rounds $\sigma^{\mathrm{a}}_p(1), \ldots, \sigma^{\mathrm{a}}_p(p)$), then v stops to participate in the simulation of $\mathcal{A}^{\mathrm{a}}$, and in particular does not write to its output register ρ in the remainder of phase p.

The first adjustment immediately ensures that the resulting algorithm $\mathcal{A}$ is indeed a WO-algorithm. We argue that $\mathcal{A}$ reaches a ready configuration in phase t, and that the output of $\mathcal{A}$ is o^{a} (and therefore correct). In phases $p < t$ there is at least one node v that is sad or did not produce an output under algorithm $\mathcal{A}^{\mathrm{a}}$, and therefore v does not become ready in the second half of phase p. In phases $p \geq t$ on the other hand, all nodes are happy and the simulation of $\mathcal{A}^{\mathrm{a}}$ corresponds to η^{a}.

Since $\mathcal{A}$ is a WO-algorithm we need to ensure that the output $o_{\mathcal{A}}$ of $\mathcal{A}$ satisfies $o_{\mathcal{A}}(v) = o^{\mathrm{a}}(v)$ for all nodes v since a node that wrote to its output register in some phase $p < t$ cannot revoke its output in later phases. Consider some node v and denote by p the phase in which v writes to its output register. This occurs in round $s = \sigma^{\mathrm{a}}_p(s^{\mathrm{a}})$ dedicated to simulating round s^{a} of $\mathcal{A}^{\mathrm{a}}$. All nodes u in the inclusive s^{a}-hop neighborhood $\Gamma^{+}_{r^{\mathrm{a}}}(v)$ must be happy in phase p (otherwise v would have received a sadness message). Moreover, the simulation that a node u at distance $d < s^{\mathrm{a}}$ from v performs of $\mathcal{A}^{\mathrm{a}}$ agrees with η^{a} for the first $s^{\mathrm{a}} - d$ rounds. Therefore for node v, the first s^{a} rounds of $\mathcal{A}$'s simulation of $\mathcal{A}^{\mathrm{a}}$ correspond to the first s^{a} rounds of η^{a}. Since $\mathcal{A}^{\mathrm{a}}$ and $\mathcal{A}^{\mathrm{r}}$ are both correct WO-algorithms, this implies that $o_{\mathcal{A}}(v) = o^{\mathrm{a}}(v)$, which concludes our argument. Lemma 6 follows.

Lemma 6 (Consecutive WO Execution)**.** *Let* Π_{C} *be a problem in* WO *and let* $\mathcal{A}^a$ *be a* WO*-algorithm solving an arbitrary problem* Π *with access to an apply-once* Π_{C}*-oracle. Then* Π *is solvable by a* WO*-algorithm without access to any oracle.*

When trying to extend the proof of Lemma 5 in attempt to establish the RW case of Theorem 3, the issue we needed to solve for RW-algorithms with an access to an apply-once oracle multiplies: Between every two simulated rounds of $\mathcal{A}^{\mathrm{a}}$, one invocation of $\mathcal{A}^{\mathrm{r}}$ is required to replace the oracle access, and a simulating node cannot know for sure that an output obtained from $\mathcal{A}^{\mathrm{r}}$ is part of a ready configuration for any such simulation of $\mathcal{A}^{\mathrm{r}}$. However, the ideas used to prove Lemma 5 can be extended to cope with this difficulty. We will show how to interleave single rounds in the simulation of an algorithm $\mathcal{A}^{\mathrm{a}}$ accessing an oracle with executions of an algorithm $\mathcal{A}^{\mathrm{r}}$ that replaces the oracle.

Proof (of Theorem 3)**.** Let C be either WO or RW, let Π_{C} be a problem in the class C, and let $\mathcal{A}^{\mathrm{r}}$ be a C-algorithm solving Π_{C}. Let $\mathcal{A}^{\mathrm{a}}$ be a C-algorithm solving an arbitrary problem Π with access to a Π_{C}-oracle (applied in every round of $\mathcal{A}^{\mathrm{a}}$). If C = RW, then by Theorem 1 and Lemma 4, we assume that $\mathcal{A}^{\mathrm{r}}$ and $\mathcal{A}^{\mathrm{a}}$ sustainably solve Π_{C} and Π, respectively. We wish to simulate $\mathcal{A}^{\mathrm{a}}$ and multiple invocations of $\mathcal{A}^{\mathrm{r}}$ using a C-algorithm $\mathcal{A}$ without access to any oracle. Denote by ρ the output register of node v. The construction of algorithm $\mathcal{A}$ is similar to the construction we used in the proofs of Lemmas 5 and 6; the difference is that in phase p, algorithm $\mathcal{A}$ should now simulate p invocations of algorithm $\mathcal{A}^{\mathrm{r}}$, one before each round of the simulated execution of $\mathcal{A}^{\mathrm{a}}$, instead of just a single invocation. That is, we precede each round $1 \leq i \leq p$ of $\mathcal{A}^{\mathrm{a}}$'s simulated execution under $\mathcal{A}$ with a simulation of an invocation of $\mathcal{A}^{\mathrm{r}}$ that runs for p rounds and replaces $\mathcal{A}^{\mathrm{a}}$'s access to the Π_{C}-oracle between rounds $i-1$ and i. Specifically, phase p now consists of $p^2 + p$ rounds, where each round $r \equiv 0 \pmod{p+1}$ of phase p is dedicated to simulating round $r/(p+1)$ of $\mathcal{A}^{\mathrm{a}}$, whereas each round $r \not\equiv 0 \pmod{p+1}$ is dedicated to simulating round $r \pmod{p+1}$ in invocation $i = \lceil r/(p+1) \rceil$ of $\mathcal{A}^{\mathrm{r}}$, occurring between rounds $i-1$ and i of $\mathcal{A}^{\mathrm{a}}$. For convenience, let σ^{a}_p denote the sequence of rounds (of $\mathcal{A}$'s execution) that are dedicated to simulating algorithm $\mathcal{A}^{\mathrm{a}}$ in phase p, i.e., $\sigma^{\mathrm{a}}_1 = \langle 2 \rangle, \sigma^{\mathrm{a}}_2 = \langle 5, 8 \rangle$ and so on.

During phase p, it may happen that the simulation of invocation $1 \leq i \leq p$ of $\mathcal{A}^{\mathrm{r}}$ in node v outputs ε, which means that v cannot simulate round i of $\mathcal{A}^{\mathrm{a}}$; when this happens, node v becomes sad for the current phase p. Recall that this means that v stops participating in the remainder of phase p and sets $\rho \leftarrow \varepsilon$. Moreover, if C = WO, then in addition to that, v broadcasts a sadness message. As before, node v sets $\rho \leftarrow \varepsilon$ during simulations of $\mathcal{A}^{\mathrm{r}}$, and when v is happy ρ is used to simulate the output register of $\mathcal{A}^{\mathrm{a}}$.

For the sake of the analysis we inductively define executions η^{r}_i of algorithm $\mathcal{A}^{\mathrm{r}}$ and an execution η^{a} of algorithm $\mathcal{A}^{\mathrm{a}}$. Let η^{r}_1 be the execution of algorithm $\mathcal{A}^{\mathrm{r}}$ that corresponds to the simulation that $\mathcal{A}$ performs to replace $\mathcal{A}^{\mathrm{a}}$'s first oracle access, and denote by o^{r}_1 the output obtained from the stable ready configuration of η^{r}_1. Both η^{r}_1 and o^{r}_1 are well-defined since under $\mathcal{A}$, the simulation of $\mathcal{A}^{\mathrm{r}}$ reuses the same random bits in every phase and due to the sustainable correctness of $\mathcal{A}^{\mathrm{r}}$. Let $\eta^{\mathrm{a}}_{(1)}$ be the first round of $\mathcal{A}^{\mathrm{a}}$'s execution η^{a} that algorithm $\mathcal{A}$ simulates in which the first oracle access of $\mathcal{A}^{\mathrm{a}}$ is replaced with o^{r}_1. Based on η^{r}_1, the first round $\eta^{\mathrm{a}}_{(1)}$ in η^{a} is well-defined. We define the executions η^{r}_i and the remaining rounds of η^{a} inductively: (1) Let η^{r}_i be the execution of algorithm $\mathcal{A}^{\mathrm{r}}$ that corresponds to the simulation that $\mathcal{A}$ performs to replace $\mathcal{A}^{\mathrm{a}}$'s oracle access after round $i-1$ of η^{a}, and denote by o^{r}_i the output obtained from the stable ready configuration of η^{r}_i. (2) Let $\eta^{\mathrm{a}}_{(i)}$ be the ith round in the execution of $\mathcal{A}^{\mathrm{a}}$ that corresponds to the simulation performed by algorithm $\mathcal{A}$ in which $\mathcal{A}^{\mathrm{a}}$'s oracle access is replaced by o^{r}_i. Note that (1) and (2) together are well-defined, since the induction is based on $\eta^{\mathrm{a}}_{(1)}$ and $\eta^{\mathrm{r}}_{(1)}$, and the simulations of $\mathcal{A}^{\mathrm{r}}$ and $\mathcal{A}^{\mathrm{a}}$ reuse the same random bits in every phase. Thanks to the sustainable correctness of $\mathcal{A}^{\mathrm{a}}$ we denote by o^{a} the output obtained from the stable ready configuration η^{a} reaches.

With these definitions in mind, denote by t^{a} the first round in which η^{a} is in a ready configuration. Denote by t^{r}_i the first round in which η^{r}_i is in a ready configuration and let $t^{\mathrm{r}} = \max_{i<t^{\mathrm{a}}}\{t^{\mathrm{r}}_i\}$. Lastly, denote

by o^{a} the output obtained from the stable ready configuration reached by η^{a} in round t^{a}. We argue that algorithm $\mathcal{A}$ reaches the first ready configuration in phase $t = \max\{t^{\mathrm{a}}, t^{\mathrm{r}}\}$, specifically in round $\sigma^{\mathrm{a}}_t(t^{\mathrm{a}})$, and that the output of $\mathcal{A}$ in that phase is o^{a}.

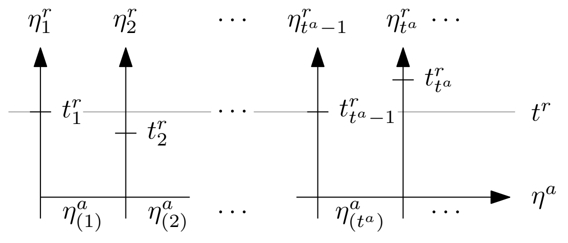

Figure 1: Executions η^{a} and η^{r}_i of $\mathcal{A}^{\mathrm{a}}$ and $\mathcal{A}^{\mathrm{r}}$, respectively.

Assume for the sake of contradiction that in some phase $p < t$ algorithm $\mathcal{A}$ reaches a ready configuration. Due to our construction this can only occur in a round $s = \sigma^{\mathrm{a}}_p(s^{\mathrm{a}})$ dedicated to simulating some round s^{a} of algorithm $\mathcal{A}^{\mathrm{a}}$. In that case all nodes are happy in round s, which can only occur if $t^{\mathrm{r}}_i \leq p$ for all $i < s^{\mathrm{a}}$. This implies that the first s^{a} rounds that $\mathcal{A}$ simulated of algorithm $\mathcal{A}^{\mathrm{a}}$ correspond to the first s^{a} rounds of η^{a}, i.e., η^{a} reaches a ready configuration in round $s^{\mathrm{a}} = t^{\mathrm{a}}$ in contradiction with the choice of t. In phase $p = t$ on the other hand, for all $i \leq t^{\mathrm{a}}$ execution η^{r}_i reaches a ready configuration within p rounds. Therefore the first t^{a} rounds that $\mathcal{A}$ simulates of $\mathcal{A}^{\mathrm{a}}$ correspond to the first t^{a} rounds of η^{a} and $\mathcal{A}$ reaches a ready configuration in round t^{a}. In the case C = RW, the (at least loose) correctness of $\mathcal{A}$ now follows from the correctness of $\mathcal{A}^{\mathrm{a}}$ and the proof is concluded by applying Theorem 1.

For the case C = WO however, we need to ensure that the output $o_{\mathcal{A}}$ of algorithm $\mathcal{A}$ satisfies $o_{\mathcal{A}}(v) = o^{\mathrm{a}}(v)$ for all nodes v, since in algorithm $\mathcal{A}$ a node v may write to its output register ρ prior to phase t. Let v be a node that irrevocably sets $\rho \leftarrow o_{\mathcal{A}}(v)$ in phase p. This can only occur in round $s = \sigma^{\mathrm{a}}_p(s^{\mathrm{a}})$ for some s^{a}. Since v did not receive a sadness message for phase p all nodes u in the inclusive s^{a}-hop neighborhood $\Gamma^+_{s^{\mathrm{a}}}(v)$ of v are happy in round s. In other words, all nodes u are ready in the first s^{a} simulations of $\mathcal{A}^{\mathrm{r}}$ performed by $\mathcal{A}$ in phase p. It follows that for node v the first s^{a} rounds of the s^{a} simulations of $\mathcal{A}^{\mathrm{a}}$ correspond to the first s^{a} rounds of η^{a}. Since $\mathcal{A}^{\mathrm{r}}$ and $\mathcal{A}^{\mathrm{a}}$ are both WO-algorithms we conclude that indeed $o_{\mathcal{A}}(v) = o^{\mathrm{a}}(v)$. □

Now that Theorem 3 is established we introduce the concept of hard problems by borrowing the terminology from sequential complexity theory.

Definition (Hardness)**.** For two classes B ⊇ C, a problem Π is said to be B-*hard* with respect to C, denoted by $\Pi \in$ *B-hard*$_{\mathrm{C}}$, if for every problem $\Pi_{\mathrm{B}} \in \mathrm{B}$, there exists a C-algorithm that solves Π_{B} with access to a Π-oracle. We say that Π is *complete* in B with respect to C, denoted by $\Pi \in$ *B-complete*$_{\mathrm{C}}$, if additionally Π itself is contained in B.

Following our notational convention, we would refer to an $\mathcal{NP}$-hard problem as being $\mathcal{NP}$-*hard*$_{\mathcal{P}}$. For example, the problem of electing a leader is well known to be CF-*hard*$_{\mathrm{WO}}$ since once a leader is available, this leader can assign unique identifiers to all other nodes and solve the problem centrally. Our definition yields the three hardness classes CF-*hard*$_{\mathrm{RW}}$, CF-*hard*$_{\mathrm{WO}}$ and RW-*hard*$_{\mathrm{WO}}$, allowing us to study how algorithms running in anonymous networks relate to centralized algorithms as well as how the two output revocability notions relate among each other. By definition, every CF-*hard*$_{\mathrm{WO}}$ problem is both CF-*hard*$_{\mathrm{RW}}$ and RW-*hard*$_{\mathrm{WO}}$; it turns out that the converse direction is also true. In Section 5 we will have the necessary tools to prove this statement, as cast in the following theorem.

Theorem 7. *The hardness classes satisfy*

$$\text{CF-}\mathit{hard}_{\mathrm{WO}} = \text{CF-}\mathit{hard}_{\mathrm{RW}} \cap \text{RW-}\mathit{hard}_{\mathrm{WO}}.$$

4 Problem Zoo

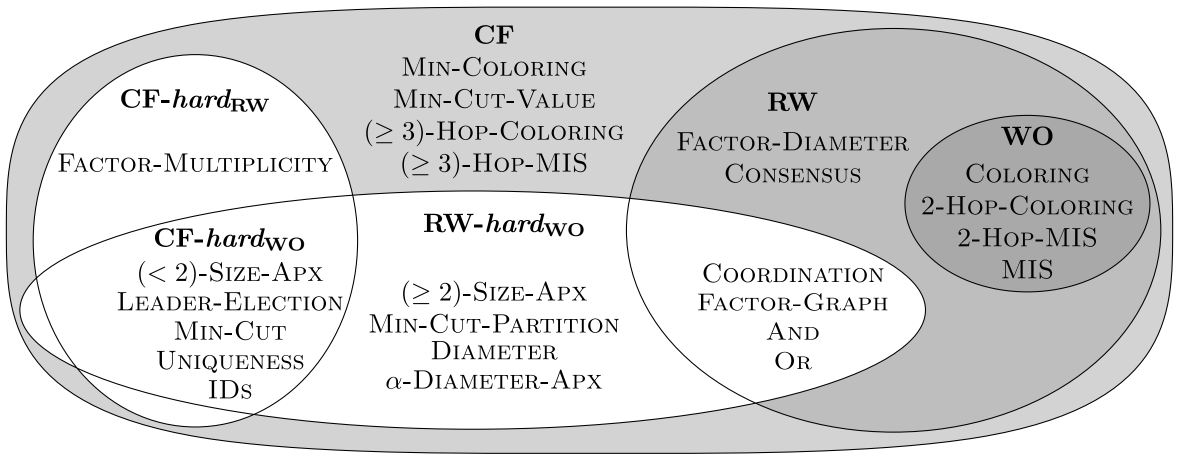

Figure 2: Classes CF, RW and WO, and the respective hardness classes.

In this section, we study the computability and hardness of various problems in our setting. A total of 21 problems are investigated as depicted in Figure 2, including variations of approximation guarantees or output specification. First, we will focus on the computability of each problem, i.e., whether it is in WO, in RW \ WO, or in CF \ RW. Later in Section 4.2, we will investigate the hardness of each of the problems. Based on that, we establish Theorem 7 in Section 5.

4.1 Computability

Almost all results regarding (non-)computability of problems derived in this section are obtained using one of two general proof frameworks. To characterize problems that can be solved by RW-algorithms, we find a necessary and sufficient condition. For the class WO, we use a necessary condition that allows us to rule out the inclusion of problems in this class. All but one result on non-computability can then be derived using the two characterizations. For computability of problems in RW, the same characterization can be used, while for problems in WO we refer to known algorithms.

4.1.1 Graph Factors, Products and Local Views

The key to our characterization of problems in RW is the notion of *graph factors*.[3]

Definition (Graph Factors)**.** Let G and H be two simple undirected graphs and ℓ_G and ℓ_H two labelings of G and H, respectively, such that ℓ_G and ℓ_H share the same co-domain. A surjective function $f : V(G) \to V(H)$ is called a *factorizing map* of G inducing H if it has the following properties:

1. if $(u, v) \in E(G)$, then $(f(u), f(v)) \in E(H)$ for every $u, v \in V(G)$, that is, f is a graph homomorphism;
2. for every node $v \in V(G)$, the restriction $f|_{\Gamma(v)}$ of f to v's neighborhood is a bijection onto the neighborhood $\Gamma(f(v))$ of v's image $f(v)$, that is, f is *locally* one-to-one and onto; and
3. the labeling functions satisfy $\ell_G(v) = \ell_H(f(v))$ for every node $v \in V(G)$, that is, f preserves the labels.

If there exists such a factorizing map f, then we say that (G, ℓ_G) is a *product* of (H, ℓ_H) or equivalently, that (H, ℓ_H) is a *factor* of (G, ℓ_G). A labeled graph (G, ℓ_G) is *prime* if all factors of (G, ℓ_G) are isomorphic, i.e., if the only factor of (G, ℓ_G) is the graph itself.

The above definition essentially corresponds to the definition given in [24] for covering graphs extended to respect node labels. It is a known fact that $|V(G)|$ must be an integer multiple m of $|V(H)|$ (see, e.g.,

[3]In the distributed computing literature, the concept of graph factors was also referred to as covering graphs and graph lifts.

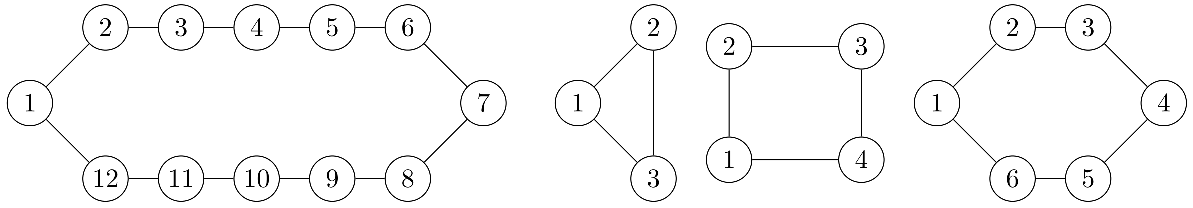

Figure 3: The cycles C_3, C_4 and C_6 on 3, 4 and 6 nodes are factors of the 12-cycle C_{12} by mapping node i in C_{12} to the node $i \pmod{3, 4 \text{ or } 6}$ in the respective cycle. The prime factors of C_{12} are C_3 and C_4.

[24]). We say that (G, ℓ_G) is an *m-product* of (H, ℓ_H) or equivalently that (H, ℓ_H) is an *m-factor* of (G, ℓ_G), denoted $(G, \ell_G) \cong m \cdot (H, \ell_H)$ and $m \cdot (H, \ell_H) \cong (G, \ell_G)$, respectively, when we want to emphasize the specific value of m. It will be convenient to use the notation $(G, \ell_G) \cong m \cdot (H, \ell_H)$ without explicitly specifying m as well, in which case the exact value of m is typically not important. Note however that an m-product of a graph is not necessarily unique (not even for $m = 1$). For two unlabeled graphs G and H, we assume that ℓ_G and ℓ_H both assign the same label to all nodes, and we omit the labeling functions in our notation.

We will use factors of graphs to derive a characterization for problems with input and output labelings i and o of a graph G, respectively. Note that the combined labeling (i, o) is also a labeling of G in which every node v is labeled by the pair $(i(v), o(v))$. If (G, i) is an m-product of (G, i') by a factorizing map f and o is a valid output labeling of (G, i), then we denote the labeling $o'(\cdot) = o(f(\cdot))$ as the *natural extension* of o to (G', i'). Observe, that in this case $(G', i', o') \cong m \cdot (G, i, o)$.

Product graphs are used in the existing literature to derive negative results for computability of problems by anonymous distributed algorithms, dating back to the seminal work of Angluin [3]. Those proofs are based on *lifting* a computation that occurs in a graph (G, i) to some product $(G', i') \cong m \cdot (G, i)$ and forcing node $v' \in V(G')$ to copy the execution of its image under the factorizing map f. This technique was used, for example, to prove the impossibility of electing a leader in anonymous networks [3], and the same technique can be used to show that Leader-Election is not in RW. As it turns out, graph products actually lead to a complete characterization of problems in RW.

Theorem 8 (Characterization of RW)**.** *Problem* Π *is in* RW *if and only if*

$$\begin{aligned}
&\forall (G, i) \in \Pi,\ \exists o : (G, i, o) \in \Pi \text{ s.t.} \\
&\forall (G', i') \in \Pi,\ \exists o' : (G', i', o') \in \Pi \text{ s.t.} \\
&(G', i') \cong m \cdot (G, i) \implies (G', i', o') \cong m \cdot (G, i, o)\,.
\end{aligned} \tag{1}$$

Consider a problem Π whose input instances are arbitrary labeled graphs with $O(\Pi) = \{\text{YES}, \text{NO}\}$, and fix some subset Y of the input instances. The problem Π is is called a *(distributed) decision problem* (cf. [31, 25]), if for every $(G, i) \in Y$, all nodes must output YES and for every input instance $(G, i) \notin Y$, at least one node outputs NO. The instances in the set Y are referred to as the *YES-instances* of Π. Theorem 8 implies that the class of decision problems in RW is exactly the class of decision problems that are *closed* under taking products of the solved problem instances, namely if $(G, i, o) \in \Pi$ and $(G', i', o') \cong m \cdot (G, i, o)$, then $(G', i', o') \in \Pi$.

The proof of Theorem 8 relies in part on the aforementioned lifting technique [3]. More specifically, fix some instance (G, i) and let (G', i') be a product of that instance by the factorizing map f. For every node $v \in V(G)$, let $\eta(v)$ denote the execution of an algorithm $\mathcal{A}$ that is invoked on (G, i) from the perspective of v. Note that η is fully determined by the random bits used by each node in the course of $\mathcal{A}$'s execution. Denote by $\eta'(\cdot) := \eta(f(\cdot))$ the natural extension of η to (G', i'). In η' every node v will perform exactly the same execution as its image $f(v)$, and if an output o is reached in execution η of $\mathcal{A}$, then the output $o'(\cdot) = o(f(\cdot))$, i.e., the natural extension of o to (G', i'), is reached in execution η. We shall refer to execution η' as *lifting* η from (G, i) to (G', i') and conclude with the following lemma.

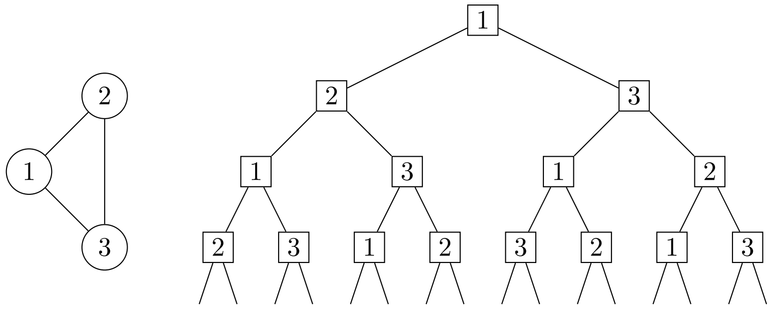

Figure 4: Cycle on 3 nodes and the corresponding local view of depth 4 as seen by node 1.

Lemma 9 (Lifting an Execution [3])**.** *Consider some* RW*-algorithm $\mathcal{A}$ and let (G, i) and (G', i') be two labeled graphs satisfying $(G', i') \cong m \cdot (G, i)$ with factorizing map $f : V(G') \to V(G)$. For every finite execution η of $\mathcal{A}$ on (G, i) ending in a ready configuration with output o, there exists a finite execution η' of $\mathcal{A}$ on (G', i') ending in a ready configuration with output o' such that $o'(v) = o(f(v))$ for every $v \in V(G')$.*

In particular, if no valid output labeling for (G, i) can be naturally extended to a valid output labeling for (G', i'), then it is also not possible for an algorithm to (always) return a correct output in both graphs. Theorem 8 is also closely related to the Factor-Graph problem introduced in Section 4.1.2 and therefore deferred until then. A necessary condition for problems in WO can be defined using *local views*.

Definition (Local View)**.** Consider some randomized algorithm $\mathcal{A}$. Let (G, ℓ) be a labeled graph and let v be a node in $V(G)$. Fix some assignment β of random bits to the nodes and denote by $\beta_t(v)$ the (finitely many) random bits used by v in all rounds $r \leq t$. The *depth-t local view* of v under β is the rooted tree $L_t^\beta(v)$ of depth t with a labeling ℓ_t defined as follows. For every node v, the local view $L_0^\beta(v)$ contains only a single vertex[4] $\mathfrak{r}$ and the labeling $\ell_0(\mathfrak{r})$ is $(\ell(v), \deg(v), \beta_0(v))$. From the labeled forest $F_t(v) := \{L_t^\beta(u) \mid u \in \Gamma(v)\}$, the depth-$(t+1)$ local view $L_{t+1}^\beta(v)$ is constructed in two steps: (1) Prune the sub-tree corresponding to node v from the root vertex $\mathfrak{r}_u$ of every $L_t^\beta(u)$, $u \in \Gamma(v)$, to obtain the pruned local view $L_t'^\beta(u)$; let $F_t'(v) = \{L_t'^\beta(u) \mid u \in \Gamma(v)\}$ be the forest containing the pruned local views of v's neighbors. (2) Construct $L_{t+1}^\beta(v)$ from the pruned local views in $F_t'(v)$ by introducing a new root $\mathfrak{r}$ as the parent of $\mathfrak{r}_u$ for all $u \in \Gamma(v)$. The labeling $\ell_{t+1}(\mathfrak{r}) := (\ell(v), \deg(v), \beta_{t+1}(v))$, whereas for all nodes in the pruned sub-trees $L_t'^\beta(u)$ of $\mathfrak{r}$, the labeling remains unchanged. In cases where no assignment of random bits is assumed the *(deterministic) depth-t local view* $L_t(v)$ is obtained in the same way by excluding $\beta_t(v)$ in the vertex labels.

Informally, the depth-t local view of node v captures the network from v's point of view in round t. Local views without random bits were used before, e.g., to discuss solvability of leader election in the context of deterministic anonymous algorithms [42]. Theorem 10 relies on the possibility that nodes whose executions are indistinguishable from v's perspective under deterministic algorithms may remain indistinguishable from v's perspective for a finite amount of time also under randomized algorithms.

Theorem 10. *Problem* Π *is not in* WO *if*

$$\begin{aligned}
&\exists (G, i) \in \Pi \text{ s.t. } \forall o : (G, i, o) \in \Pi,\ \forall t \in \mathbb{N},\ \exists (G', i') \in \Pi \text{ s.t.} \\
&\forall o' : (G', i', o') \in \Pi,\ \exists v \in G,\ \exists v' \in G' \text{ s.t.} \\
&L_t(v) = L_t(v'), \text{ and} \qquad (2)\\
&o(v) \neq o'(v') \,. \qquad (3)
\end{aligned}$$

[4] To avoid the confusion between the basic elements in the graph G and those in the rooted tree $L_t^\beta(v)$, we refer to the former as nodes and to the latter as vertices.

Proof. Let $W(\Pi)$ denote the characterization for a problem Π stated in the theorem. Assume for the sake of contradiction that there exists a problem $\Pi \in \text{WO}$ for which $W(\Pi)$ holds and let $\mathcal{A}$ be a WO-algorithm solving Π. Invoke $\mathcal{A}$ on the input instance (G, i) promised by $W(\Pi)$ to obtain $\mathcal{A}$'s output o after t steps and denote the random bits used by node v up to round t in this execution of $\mathcal{A}$ by $\beta_t(v)$. Let $(G', i') \in \Pi$ be the labeled graph promised by $W(\Pi)$ for (G, i, o) and t. For every valid output o' to (G', i'), the property $W(\Pi)$ guarantees the existence of two nodes $v \in V(G)$ and $v' \in V(G')$ satisfying both (2) and (3). Constraint (2) implies that with positive probability nodes v and v' observe the same execution up to (and including) round t, namely if $L_t^{\beta}(v) = L_t^{\gamma}(v')$ for some assignment of random bits γ to nodes in G'. Therefore, with positive probability, v' will return an output $o'(v') = o(v)$. But (3) implies that o' cannot be a valid output for (G', i'), in contradiction to the assumption that algorithm $\mathcal{A}$ solves Π. ☐

4.1.2 Results

We start by briefly stating the proof techniques derived from Theorems 8 and 10 that we use to establish computability results.

$\Pi \notin$ WO: The inclusion of Π in WO will be disproved by finding an input instance $(G, i) \in \Pi$ and for all valid outputs to (G, i) and arbitrary t, a construction of an input instance $(G', i') \in \Pi$ in which the depth-t local view of some node $v' \in V(G')$ is the same as that of some node $v \in V(G)$, but the output of v' must differ from that of v.

$\Pi \notin$ RW: The inclusion of Π in RW will be disproved by finding an input instance $(G, i) \in \Pi$ and for all valid outputs o to (G, i), an input instance $(G', i') \in \Pi$ satisfying $(G', i') \cong m \cdot (G, i)$ such that no natural extension of o to (G', i') is a valid output for that instance.

$\Pi \in$ RW: The inclusion of Π in RW will be established by showing that for every input instance $(G, i) \in \Pi$, there is a valid output o such that for every input instance $(G', i') \in \Pi$ satisfying $(G', i') \cong m \cdot (G, i)$, the natural extension of o to (G', i') is a valid output for that instance.

The two techniques for RW rely on Theorem 8, which we did not prove yet. Therefore, after giving a brief overview of problems known to be in WO, we will focus on proving the theorem first.

MIS and other Local Symmetry Breaking. The well studied symmetry breaking tasks MAXIMAL-INDEPENDENT-SET (MIS), $(\Delta + 1)$-COLORING and MAXIMAL-MATCHING are indeed in WO: The famous Luby-algorithm [35, 2] satisfies the WO condition already. Similarly, there are algorithms to solve $(\Delta + 1)$-COLORING [33][5] and MAXIMAL-MATCHING [29] that are WO-algorithms. Two other problems studied before are 2-HOP-MIS and 2-HOP-COLORING in which two nodes in the independent set or two nodes having the same color, respectively, must not have a common neighbor. In [17] both problems were found solvable by WO-algorithms using an even weaker computational model. The algorithm from [17] that solves 2-HOP-COLORING uses up to $\Delta^2 - \Delta + 1$ colors, which is a simple upper bound on the number of required colors.

Factor-Graph. In the FACTOR-GRAPH problem, nodes in the network (G, i) are required to agree on a factor (H, j) of (G, i). That is, every node $v \in G$ should output the same factor (H, j) of (G, i) (with inputs and uniquely named nodes), and its own name $f(v)$ in H, where f is the factorizing map inducing H. Had we proven Theorem 8 already, it would follow from the definition that FACTOR-GRAPH it is in RW. Instead we use this problem to establish the theorem, starting with the following observation which is essential for the first half of the proof.

Lemma 11. *There is a* RW*-algorithm solving* FACTOR-GRAPH*.*

Proof. We present a RW-algorithm $\mathcal{A}$ that solves FACTOR-GRAPH on arbitrary input instances (G, i). Algorithm $\mathcal{A}$ progresses in phases where during each phase p, every node v constructs a *candidate factor* (G_p, i_p). Nodes in $V(G_p)$ are identified by a randomly chosen (candidate) *identifier* $\beta_p(v)$, and an edge

[5]The algorithm for $(\Delta+1)$-COLORING described in the cited work also works if no upper bound on Δ is known by replacing a node of degree d in the overlay graph with a complete graph on $d + 1$ nodes.

$\{\beta_p(u), \beta_p(v)\}$ is added to $E(G_p)$ if the edge $\{u, v\}$ is present in $E(G)$. All nodes $v \in V(G)$ start in phase 1, and advance from phase p to $p+1$ if v sends or receives an *inhibiting message* for phase p.

In the beginning of a phase p, all nodes v first choose a random bit string $\beta_p(v)$ containing p random bits. Node v then exchanges $\beta_p(v)$, its input $i(v)$, and its degree $\deg(v)$ with every neighbor. After v received a message containing the corresponding values of every neighbor, it broadcasts a *my-neighborhood* message $M_p(v)$ containing $(\beta_p(v), \deg(v), i(v))$, and the corresponding values of all its neighbors. While v receives my-neighborhood messages $M_p(u)$ from other nodes u, node v gradually constructs its candidate factor (G_p, i_p) by inserting the node $\beta_p(u)$ with the label $i(u)$ contained in $M_p(u)$, and edges to all of u's neighbors. Note that some edges may point to nodes that were not yet inserted into the graph. We say that v detects an *inconsistency*, if either two messages $M_p(u) \neq M_p(u')$ are received for which $\beta_p(u) = \beta_p(u')$, or if a message from a node u with degree $\deg(u)$ was received that did not contain $\deg(u)+1$ different identifiers for u and its neighbors. When v detects an inconsistency it broadcasts an inhibiting message for phase p. A node v sending an inhibiting message for the current phase p sets its output register to ε and starts phase $p+1$. If v did not receive an inhibiting message for a phase p and all endpoints of edges in (G_p, i_p) were inserted, then v returns the output $((G_p, i_p), \beta_p(v))$.

We start the analysis of algorithm $\mathcal{A}$ by showing that $\mathcal{A}$'s output is correct if a ready configuration is reached. For this, observe that if two neighboring nodes u and v are in different phases p_u and p_v respectively, then u or v is currently broadcasting an inhibiting message and is therefore not ready. When on the other hand all nodes are in the same phase p and all nodes are ready, then no node detected an inconsistency in (G_p, i_p). Therefore the returned graph (G_p, i_p) is the same graph for every node, and we have to show that β_p is a factorizing map inducing (G_p, i_p). The function β_p is surjective, because every node in $V(G_p)$ has a preimage in G. Further β_p is a graph homomorphism since for every edge $\{u, v\}$ in G the edge $\{\beta_p(u), \beta_p(v)\}$ is inserted into G_p. The inconsistency detection ensures that the restriction $\beta_p|_{\Gamma(v)}$ is an injection on $\Gamma(\beta_p(v))$ for every node v. Because the input labeling $i_p(\beta_p(v))$ is defined by the input value assigned to v, the function β_p respects the graph labeling, and we conclude that $m \cdot (G_p, i_p) \cong (G, i)$ for some m. It is left to show that $\mathcal{A}$ reaches a ready configuration with probability 1. But this will happen at latest in a phase p_0 in which every node chooses a unique random identifier, because this ensures that every my-neighborhood message is unique. In this case the algorithm will return a graph G_{p_0} that is isomorphic to G. $\square$

Having established that Factor-Graph is a problem in RW, we now present the proof of Theorem 8.

Proof (of Theorem 8)**.** Let $R(\Pi)$ denote the graph theoretic characterization (1) stated in the theorem, that is

$$\begin{aligned} R(\Pi) = &\forall (G, i) \in \Pi, \exists o : (G, i, o) \in \Pi \text{ s.t.} \\ &\forall (G', i') \in \Pi, \exists o' : (G', i', o') \in \Pi \text{ s.t.} \\ &(G', i') \cong m \cdot (G, i) \implies (G', i', o') \cong m \cdot (G, i, o). \end{aligned}$$

We wish to prove that $\Pi \in \text{RW} \Leftrightarrow R(\Pi)$, and we prove both directions of the if and only if separately.

if: Let Π be a distributed problem that satisfies $R(\Pi)$; we prove that then Π must be in RW. To accomplish that, we describe a RW-algorithm $\mathcal{A}$ solving Π with access to a Factor-Graph-oracle. Since Factor-Graph is solvable by a RW-algorithm without access to any oracle (Lemma 11) and oracles are sound (Theorem 3), this is sufficient to conclude that $\Pi \in \text{RW}$. The key to algorithm $\mathcal{A}$ is to invoke the Factor-Graph-oracle until returns a valid input instance (G, i) of Π. For every such instance, the characterization $R(\Pi)$ promises the existence of a valid output o to (G, i) satisfying that for every product $(G', i') \cong m \cdot (G, i)$, with $(G', i') \in \Pi$, the natural extension o' of o to (G', i') is a valid output for (G', i'). Algorithm $\mathcal{A}$ exploits that as follows.

Fix some instance $(G, i) \in \Pi$. At the beginning of round r, node v appends a random bit to the (initially empty) string $\beta_{r-1}(v)$ to obtain $\beta_r(v)$. Then, node v invokes the oracle with input $(i(v), \beta_r(v))$. In all rounds $r > 1$ the oracle output register of every node v contains a labeled graph $(H_r, (j_r, \gamma_r))$ satisfying $(H_r, (j_r, \gamma_r)) \cong m \cdot (G, (i, \beta_{r-1}))$, and every node receives a name $f_r(v) \in V(H_r)$ assigned to v by the factorizing map inducing H_r. Node v now checks whether (H_r, j_r) is an input instance of Π. If it is, then v

chooses the lexicographically smallest o_r that satisfies $R(\Pi)$ for (H_r, j_r) and writes $o_r(f_r(v))$ to its output register.

When in round r every node v returns some output $o_r(v)$, the output of algorithm $\mathcal{A}$ is valid for the instance (G, i) on which the algorithm is executed, because $(G, i) \cong m \cdot (H_r, j_r)$. Algorithm $\mathcal{A}$ will reach a stable ready configuration with probability 1 within finite time, since the output from the oracle will satisfy $(H_r, j_r) \cong 1 \cdot (G, i)$ in round r if every node tossed a unique random string $\beta_{r-1}(v)$ in round $r-1$. Notice that $\mathcal{A}$ does not need to change its output register once it wrote to it, which allows us to conclude that in fact Factor-Graph is in fact RW-*complete*$_{\mathrm{WO}}$.

only if: For the sake of contradiction, assume that $\neg R(\Pi)$ holds for some problem $\Pi \in$ RW, that is

$$\begin{aligned}\neg R(\Pi) = {} & \exists (G,i) \in \Pi \text{ s.t. } \forall o : (G,i,o) \in \Pi,\\ & \exists (G',i') \in \Pi \text{ s.t. } \forall o' : (G',i',o') \in \Pi :\\ & (G',i') \cong m \cdot (G,i) \wedge \neg\big((G',i',o') \cong m \cdot (G,i,o)\big).\end{aligned}$$

Let $\mathcal{A}$ be a RW-algorithm solving Π, let η be an execution of $\mathcal{A}$ on the instance (G, i) promised by $\neg R(\Pi)$, and let o be the output of $\mathcal{A}$ obtained in η. Note that o satisfies $(G, i, o) \in \Pi$, and therefore the property $\neg R(\Pi)$ guarantees the existence of some $(G', i') \in \Pi$ with $(G', i') \cong m \cdot (G, i)$ such that $(G', i', o') \cong m \cdot (G, i, o)$ does not hold for any o', i.e., the natural extension of o to (G', i') is not a valid output to (G', i'). Lift the execution η of $\mathcal{A}$ on (G, i) to obtain the execution η' of $\mathcal{A}$ on (G', i'). By Lemma 9 we see that $\mathcal{A}$'s output o' in execution η' is the natural extension of o to (G', i'), contradicting the assumption that $\mathcal{A}$ solves Π. □

As stated in the *if*-part of the proof, Factor-Graph is RW-*complete*$_{\mathrm{WO}}$, and as such cannot be solved by a WO-algorithm.

Corollary 12. *Finding a factor of the input graph is* RW-*complete*$_{\mathrm{WO}}$.

Coordination. In coordination problems nodes in the network keep track of some shared state and wish to determine whether their shared state is in unison. This kind problem arises for example in atomic commit protocols and in the two generals problem, where all participants in the network need to agree on the same value before they can proceed. More formally, we consider the problem Coordination where the input instances are all labeled graphs, and the solved instances satisfy the following. If all nodes are labeled with the same input label, then all nodes output "UNISON", otherwise there is at least one pair of nodes with different labels and all nodes output "DISCORD".

Coordination is not contained in WO. Let (G, i) be the 3-cycle with input 0, so the output to this instance is "UNISON" for all nodes in G, and let v be any node in $V(G)$. For arbitrary t, let further (H, j) be the cycle on $2t + 1$ nodes, in which exactly one node w gets input 1, and let v' be the node in $V(H)$ furthest away from w. The depth-t local views of v and v' are equal, but while node $v \in V(G)$ must return "UNISON" the only correct output of $v' \in V(G')$ is "DISCORD".

We stated that the class RW is essentially the class of coordination problems, and indeed we use the characterization of Theorem 8 to show that Coordination is in RW. For this, let (G, i) be a labeled graph in which all nodes get the same input x. In all products of (G, i) every node has input x, so returning "UNISON" leads to a correct output in all products of (G, i). On the other hand, if (G, i) contains two nodes $u \neq v$ with different inputs $x \neq y$ respectively, then all its products also contain nodes with different inputs x and y, and therefore the output "DISCORD" for all nodes can be extended to all products of (G, i). In Section 4.2 we show that Coordination is complete in RW.

Logical And & Or. The definition for the problems And and Or are straigt-forward: All nodes are provided with a binary input value and have to compute the logical conjunction resp. disjunction of all those inputs.

The problem And (Or) is not contained in WO for essentially the same reason as Coordination: Let (G, i) be the 3-cycle with input 1 (0), so the only admissible output to this instance is 1 (0), and let v be any node in $V(G)$. For arbitrary t, let further (H, j) be the cycle on $2t + 1$ nodes, in which exactly one node w

gets input 0 (1), and let v' be the node in $V(H)$ furthest away from w. The depth-t local views of v and v' are equal, but v and v' are not allowed to return the same output. The two problems are, however, both in RW (again for similar reasons as Coordination is). If an input instance (G, i) contains a node with input 0 (1 for Or), then all products of (G, i) also contain a node with that input, and the only correct output for every node is 0 (1) in these instances. If on the other hand no node in (G, i) has input 0 (1), then the only correct output in (G, i) is 1 (0) for every node, which is also true for all products of such an input instance.

Consensus. In the well-known binary Consensus problem nodes can have either 0 or 1 as possible input. All nodes are required to agree on the same output, which must also appear as input to at least one node.

Like the Coordination problem Consensus is also not solvable by a WO-algorithm, but Theorem 10 cannot be used to disprove that. Instead, assume for the sake of contradiction that there is a WO-algorithm $\mathcal{A}$ solving Consensus. Let (G_0, i_0) and (G_1, i_1) be 3-cycles, where in G_0 every node gets input 0 and in G_1 every node gets input 1. Execute $\mathcal{A}$ on both instances to obtain correct output labelings o_0 and o_1 after t_0 and t_1 rounds, respectively, and denote the random bits used in the execution of $\mathcal{A}$ up to round t on each respective instance by β_t and γ_t. Let u_0 and u_1 denote two arbitrary nodes in G_0 and G_1. Let further G' be the cycle consisting of $2 \cdot (t_0 + t_1 + 1)$ nodes, and denote by u'_0 and u'_1 two nodes in G' with maximal distance. It is possible to assign inputs and random bits δ_t to all nodes so that $L^{\beta}_{t_0}(u_0) = L^{\delta}_{t_0}(u'_0)$ and $L^{\beta}_{t_1}(u_1) = L^{\delta}_{t_1}(u'_1)$, i.e., the two nodes u'_0 and u'_1 in G' observe the same depth-t_0 and depth-t_1 local view under δ_t as the corresponding nodes u_0 and u_1 did in G_0 and G_1, respectively. Thus, there is an execution of $\mathcal{A}$ (by choosing the random bits as determined by δ_t) which leads to a configuration where the output returned by u'_0 will be 0, while that of u'_1 will be 1, contradicting our assumption.

However, we can show that Consensus is in RW by applying Theorem 8. For this, let (G, i) be a labeled graph in which all nodes get input 0. In all products of (G, i) every node has input 0, so agreeing to output 0 is valid in all products of (G, i). On the other hand, if (G, i) contains a node with input 1, then all its products also contain a node with input 1, and therefore the output in which all nodes agree on 1 can be extended to all products of (G, i).

Factor-Multiplicity. Another problem related to graph factors is Factor-Multiplicity: In an *unlabeled* network G, every node should output the multiplicity m of a graph H such that $m \cdot H \cong G$, and the number of nodes in H is minimal among all possible factors of G. The last constraint prohibits the nodes from answering 1 in every graph (each graph is of course a factor of itself).

The problem is not solvable for RW-algorithms: Let G be any prime graph, for example a 3-cycle, such that the smallest factor of G has multiplicity 1. In any non-trivial product of G, this answer is however not correct. Using this problem we will establish in Section 4.2.1 that the two hardness classes CF-$\mathit{hard}_{\mathrm{RW}}$ and CF-$\mathit{hard}_{\mathrm{WO}}$ are distinct.

Factor-Diameter. Agreeing on the diameter of some factor of an input instance is certainly possible in the RW model, as nodes able to agree on a factor, and may just output its diameter. To see that the problem is not solvable by a WO-algorithm let G be the 3-cycle, so that the only admissible outputs for G will be those in which the agreed upon factor H is a 3-cycle and all three nodes choose a different name. For every t, construct the cycle G' on p nodes where $p > t$ is prime so that in particular, G' is prime. Any arbitrarily chosen $v \in V(G)$ and $v' \in V(G')$ satisfy $L_t(v) = L_t(v')$, but the only admissible output $o'(v')$ is $\lfloor p/2 \rfloor$ for every $v' \in V(G')$. However, the two problems differ in their hardness, as we will see in Section 4.2.1.

***k*-Hop-MIS, *k*-Hop-Coloring and Min-Coloring.** In the k-Hop-MIS problem, nodes shall output a maximal set in which any two nodes in the set have at least distance k (measured in hops), i.e., a shortest path between them uses $k+1$ edges. Similarly, in a solution to the k-Hop-Coloring problem, nodes having the same color must be at least k hops apart. As we saw earlier, both problems are in WO for $k \leq 2$.

For $k > 2$ they are not in RW, and neither is coloring with the minimum amount of colors. To see this for k-Hop-Coloring let G be the triangle so that every solution to G will use exactly three different colors. Now, let $G' \cong 2 \cdot G$ be the 6-cycle. Any valid k-Hop-Coloring of G' with $k > 2$ needs to use six colors,

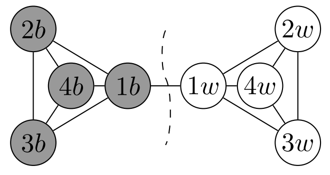

Figure 5: Graph G, with unique minimum cut 1. Black and white nodes indicate the two partitions in the output of the minimum cut.

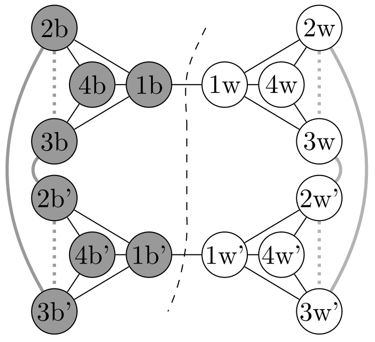

Figure 6: Graph G', which is a 2-product of G from Figure 5, with increased cut value and double the number of nodes. The dashed edges are replaced by the corresponding thick edges without violating the graph factor property.

thus the natural extension of a valid output o to G cannot be a valid output for G'. On the other hand, a Min-Coloring of the 6-cycle only needs two colors, therefore a natural extension of o to G' will not be a minimum coloring on the 6-cycle. Similarly, for k-Hop-MIS, a 2-product of a valid output on the 3-cycle violates the distance requirement on the 6-cycle.

Diameter and approximating it. We consider the problem of finding the Diameter of the network as well as the approximation problem Diameter-Apx. The problem is not in RW, which can be seen by—again—taking G to be the triangle (with diameter one), and G' to be the cycle on six nodes with diameter three. The only valid output labeling for G cannot be extended to a valid output labeling on G'. As for the approximation problem with approximation ratio α, let G' be the $3\lfloor\alpha + 1\rfloor$-cycle. We will however see in Section 4.2.1 that both problems are prime examples for the class of problems that are RW-$hard_{\mathrm{WO}}$.

Min-Cut. A surprisingly interesting problem to study is the Min-Cut problem. There are basically three ways to define the Min-Cut problem: We can require nodes to learn the *value* of the minimum cut, a *partition* of the nodes (say, into black and white nodes) inducing the minimum cut, or thirdly, we can ask for the combination of *both*. We denote those variants by Min-Cut-Value, Min-Cut-Partition and Min-Cut, respectively. This does not change their computability (no variant can be solved in an anonymous network), but we will in Section 4.2.1 find that the exact specification *does* make a difference for the hardness of each single variant.

To prove that none of the problem variants is in RW, first observe that the graph G in Figure 5 has a unique minimum cut. Therefore, every valid output to the min-cut problem in G must output the cut-value and/or partition indicated in the figure. Because the graph G' in Figure 6 is a product of G, but has a different cut value, Min-Cut-Value is not solvable in RW. From this we can immediately follow that Min-Cut can also not be solved. To see that Min-Cut-Partition is also not solvable, we alter G' slightly to obtain G'' in which only the edges {2b, 3b} and {2b', 3b'} are replaced to connect {2b, 3b'} and {2b', 3b} (as indicated in the figure), while the edges connecting $\{2w, 3w\}$ and $\{2w', 3w'\}$ are left unchanged. G'' has a cut of size 1 and is also a product of G. Thus the only valid output o to G cannot be naturally extended to obtain a valid output in every product of G.

Computing with unique identifiers: Leader-Election et cetera. In the Leader-Election problem, we demand of a valid solution that there is exactly one node with output "leader", and all other nodes return "not leader". As mentioned above, [3] showed that Leader-Election is not even solvable in the RW model. It is well understood (see, e.g., [40]) that accessing a Spanning-Tree-oracle, or an oracle that equips every node with a unique identifier (the IDs problem) is equivalent to having a single leader already for WO-algorithms. The Uniqueness problem, in which nodes have to test whether every node is supplied with a unique input and output "ALL UNIQUE" or "NOT ALL UNIQUE" depending on the outcome of this test, is CF-*complete*$_{\text{WO}}$ as well. (Nodes can find unique identifiers by invoking the Uniqueness-oracle with random strings of increasing length until it replies with "ALL UNIQUE", but it is not in RW because no solution for the 3-cycle can be extended to a solution on the 6-cycle). Similarly, knowing the network Size n can be considered equivalent in our model, because nodes can broadcast random identifiers of increasing length until they observe exactly n different identifiers. On the other hand, an approximation Size-Apx of the network size with approximation guarantee α for the triangle G is not a valid approximation on a ring of size $3\lfloor \alpha + 1 \rfloor$, convincing us that not even on cycles one can find a α-Size-Apx with a RW-algorithm. In [40] the authors present a Monte Carlo algorithm to construct a spanning tree that can be turned into a Las Vegas algorithm if a $(2-\varepsilon)$-Size-Apx is known. Using our terminology, the same can be seen by giving a RW-algorithm access to an apply-once oracle to α-Size-Apx and computing a Factor-Graph. Denote the true network size by n, the approximation provided by the oracle by $\bar{n}$ and the number of nodes in a computed factor by n_{F}. If it is guaranteed that $\bar{n} < 2 \cdot n$, then a found factor H of G is indeed G itself if and only if $n_{\text{F}} \leq 2 \cdot \bar{n}$, since $|V(G)|$ must be an integer multiple of $|V(H)|$. On the other hand, when the approximation factor $\alpha \geq 2$, the answer 6 can be supplied from the oracle in both the triangle graph as well as in the ring of six nodes, and a RW-algorithm with access to such an oracle has no means of distinguishing the two.

4.2 Hardness of Problems

The last discussion already gave us an understanding of problems that are known to be CF-*hard*$_{\text{WO}}$, and while investigating Factor-Graph we already found the problem to be RW-*complete*$_{\text{WO}}$. In determining the exact containment of each problem introduced in the last section, we find all three hardness classes CF-*hard*$_{\text{WO}}$, CF-*hard*$_{\text{RW}}$ and RW-*hard*$_{\text{WO}}$ and also the three corresponding classes of complete problems to be non-empty and distinct.

4.2.1 Results

To show a problem $\Pi \in$ B is B-*hard*$_{\text{C}}$ it will be sufficient to describe a C-algorithm that solves a complete problem for C with access to a Π-oracle. In order to fully classify each problem we are also interested in negative results regarding completeness to completely characterize each of the studied problems. The following techniques derived from Theorems 8 and 10 will be used to show that a problem to not be in one of the hardness classes.

$\Pi \notin$ CF-*hard*$_{\text{RW}}$: To prove that a problem Π does not empower RW-algorithm to solve problems in CF we start with a graph (G, i). From this, we construct an m-product $(G', i') \cong m \cdot (G, i)$ with $m > 1$. We will have to ensure that there is a sequence of oracle answers to Π supplied to nodes in (G, i) that is also a valid sequence of oracle answers to the corresponding nodes in (G', i'). This is less of a problem, if Π does not take any input (other than the topology of the graph itself), because the same oracle can be used in every round. Treating the answers supplied by the oracle as additional input labels to each node, this disproves the existence of a RW-algorithm accessing an oracle as an implication of Theorem 8.

$\Pi \notin$ RW-*hard*$_{\mathbf{WO}}$: One needs to show that there is a problem Π' in RW that cannot be solved in the WO model, even if an oracle supplies each node with a solution to Π. We find an input instance $(G, i) \in \Pi$, and for all valid outputs $(G, i, o) \in Pi$ and finite t, we describe the construction of a graph (G', i'). In the construction we will specify two nodes $v' \in V(G')$ and $v \in V(G)$, and a sequence of t oracles for each graph, such that depth-t local view of v (including the answers supplied by the oracles) is the same as that of v', but the output of v' must differ from that of v. Treating the answers supplied by the oracle as additional input labels to each node, Theorem 10 then implies that Π cannot be in WO.

We omit the previously discussed results on problems that are CF-*hard*$_{\mathrm{WO}}$ and the completeness of Factor-Graph, and start by presenting another problem that is complete in RW with respect to WO.

Coordination. Indeed, Coordination is RW-*complete*$_{\mathrm{WO}}$. We show how to turn a RW-algorithm $\mathcal{A}_{\mathrm{RW}}$ solving Π without access to an oracle into a WO-algorithm $\mathcal{A}_{\mathrm{WO}}$ that solves Π with access to an Coordination-oracle. In algorithm $\mathcal{A}_{\mathrm{WO}}$ every node v will simulate one round of $\mathcal{A}_{\mathrm{RW}}$ in every round; we denote v's simulated output register of $\mathcal{A}_{\mathrm{RW}}$ by ρ_{RW}, and the actual output register of $\mathcal{A}_{\mathrm{WO}}$ by ρ_{WO}. If in round r the register $\rho_{\mathrm{RW}} = \varepsilon$, then v writes "NOT READY" to the input register of the oracle, otherwise it invokes the oracle with input "READY". When the oracle answers "UNISON" in round $r+1$ and node v was ready in round r, the network was in a ready configuration in round r, and v sets ρ_{WO} to the value contained in ρ_{RW} in round r.

Logical And & Or. The problem And as well as Or can be used in a similar way to determine whether the network is in a ready configuration at the end of every simulated round. In the previous construction we used to show hardness of Coordination, one only needs to replace the value "NOT READY" with 0 if the simulating algorithm is accessing a And-oracle (with 1 for an Or-oracle), and the value "READY" with 1 (0) respectively.

Consensus. Consensus is not RW-*hard*$_{\mathrm{WO}}$. We prove this by showing that an oracle to Consensus cannot be used to solve Or with a WO-algorithm. Once more, let (G, i) be the 3-cycle with input 0 so the only admissible output to Or for this instance is 0. For any t, let (H, j) be the cycle on $2t+1$ nodes, in which exactly one node w gets input 1 while all other nodes get input 0. Let v be any node in $V(G)$ and denote by $\nu_r(v)$ the content stored in v's oracle input register in round $r-1$ so that the value $\nu_r(v)$ is a valid answer from the Consensus-oracle in round r for all nodes in G. Let further v' be the node in $V(H)$ with maximum distance to w. Since $L_t(v) = L_t(v')$ the value $\nu_r(v)$ is the same as $\nu_r(v')$, the value that v' provides to the oracle in round r, for all $r \leq t$. Regardless of the input of w to the oracle, the answers $\nu_r(v) = \nu_r(v')$ are also valid in (H, j) for $r \leq t$. Thus we have $L_t(v) = L_t(v')$, but $o(v) = 0$, while in a valid output labeling o', node v' must output 1. Note that the same reasoning can also be used to disprove hardness of other problems such as k-Set-Agreement.

Factor-Multiplicity. We show that the problem of finding the multiplicity of a smallest (by number of nodes) factor of an unlabeled graph is CF-*hard*$_{\mathrm{RW}}$. To establish that, we define the helper problem Π_M. The input instances $(G, i) \in \Pi_M$ are all graphs G in which the label assigned to every node by i is the multiplicity m of the smallest factor $H \cong m \cdot G$. An output labeling o is valid, if in o, exactly one node is labeled "leader" while all others are labeled "not leader". Observe that $\Pi_M \in$ RW if and only if there exists a RW-algorithm $\mathcal{A}$ that solves Leader-Election with an access to an (apply-once) Factor-Multiplicity-oracle.

We argue that Π_M is indeed in RW. To that end, we prove that Π_M fulfills the characterization of Theorem 8. Let G be some graph and let H be its smallest (prime) factor, where $G \cong m \cdot H$. By definition, the input labeling i satisfies $i(v) = m$ for every node $v \in V(G)$. Consider some graph $(G', i') \cong c \cdot (G, i)$. If $c > 1$, then $(G', i') \cong (c \cdot m) \cdot H$ and therefore, (G', i') is not an input instance of Π_M for the input labeling i' that assigns m to all nodes. Otherwise, if $c = 1$, then the factorizing map $f : V(G) \to V(H)$ is in fact an

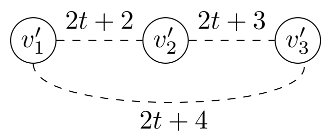

Figure 7: Counterexample showing that Factor-Diam is not RW-*hard*$_{\text{WO}}$. Dashed edges indicate a path of the denoted length, node labels indicate the mapping of nodes to the 3-cycle factor.

isomorphism and for every valid output o to (G, i), the natural extension $o(f(\cdot))$ is a valid output to (G', i'). The argument follows since every graph is either prime or a product of another prime graph.

Next, we show that Factor-Multiplicity is not RW-*hard*$_{\text{WO}}$. To see this, we argue that a WO-algorithm with access to a Factor-Multiplicity-oracle cannot solve Or. Again, let (G, i) be the triangle in which all nodes get input 0, so in a correct output to this instance all nodes will agree on 0; the only valid answer from the oracle is multiplicity 1 since the triangle is a prime graph. Now, for arbitrary t let (G', i') be the cycle on $p > 2t$ nodes where p is prime in which all but one node have input 0, and exactly one node w' gets input 1. Let v be any node in (G, i) and let v' be a node in (G', i') furthest away from w'; in particular, the distance between v' and w' is least t. Then both v and v' observe the same depth-t local view, but in a valid output to (G', i') all nodes must agree to return 1.

Factor-Diameter. As we have established the Factor-Graph problem as being RW-*hard*$_{\text{WO}}$, one might think that a WO-algorithm with access to a Factor-Diam-oracle is able to solve problems in RW. This is not the case, and thus Factor-Diam is another example of a problem in RW that is not RW-*hard*$_{\text{WO}}$. The sufficient condition stated in Theorem 10 is not strong enough to disprove this problems hardness. We will however use a very similar technique that relies on *multiple* nodes in G to "reappear" in G'.

As usual, let G be the triangle, so the only valid answer from the oracle is 1, and denote by v_1, v_2 and v_3 the three nodes in G. For any t, let G' be the ring of size $3(2t+4)$ as depicted in Figure 7. Then G is a factor of G', and therefore 1 is a valid answer of the Factor-Diam-oracle to any node in G'. The paths between the three nodes depicted in Figure 7 contain at least $2t + 2$ nodes. For every finite t, fix an assignment β_t of random bits to nodes. Because the t-hop neighborhoods of the three nodes in G' do not overlap, it is possible to find γ_t that satisfies $L_t^{\beta}(v_k) = L_t^{\gamma}(v_k')$ for $k \in \{1, 2, 3\}$. This assignment of random bits γ will occur in G' with positive probability. Because the local views of v_1, v_2 and v_3 in G' will then be the same as that in G, they will also return the same output. In particular, all will agree on the triangle as the factor, and each one will map itself do a different node in the triangle. But this means each path between v_1', v_2' and v_3' must be of length $1 \pmod 3$. This cannot be the case, because the three paths have different lengths even modulo 3, contradicting our assumption.

***k*-Hop-MIS, *k*-Hop-Coloring and Min-Coloring.** None of these problems is CF-*hard*$_{\text{RW}}$ nor RW-*hard*$_{\text{WO}}$ for any constant k. Let G be the $(2k + 1)$-cycle, and fix a solution (G, s) to k-Hop-MIS, k-Hop-Coloring or Min-Coloring to be the oracle supplied to nodes in G. To see that none of the problems is CF-*hard*$_{\text{RW}}$, let $G' \cong 2 \cdot G$ be the $4k + 2$-cycle and s' a natural extension of s to G', so s' is also a valid oracle to the chosen problem. But since G' is a 2-product of G a RW-algorithm cannot elect a leader in both G and G', not even with access to an oracle to one of the problems k-Hop-MIS, k-Hop-Coloring or Min-Coloring. We use the same graph G to show that they are not RW-*hard*$_{\text{WO}}$ by arguing why Or cannot be solved using a WO-algorithm with access to an oracle to any of the three problems. Let (G, i) be an input instance to Or on the $(2k + 1)$-cycle in which every node v in G receives input $i(v) = 0$ so that for all nodes the only valid output is $o(v) = 0$. For arbitrary t, let $(H_t, j_t) \cong t \cdot (G, i)$ be the cycle on $t \cdot (2k + 1)$ nodes in which exactly one node w receives input 1, while all other nodes v' get input 0. Because the problems do not depend on any input, we may safely assume an oracle to one of the problems supplies the same answer in every round. Denote by s_t the natural extension of s from (G, i) to (H_t, j_t) such that s_t is a valid oracle in (H_t, j_t). The node v' in (H_t, j_t) which is furthest away from w and its corresponding node v in G satisfy $L_t(v) = L_t(v')$, even when taking s_t into account, and while the only valid output of v

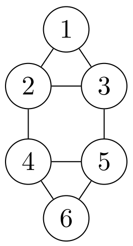

Figure 8: Graph G with diameter 3.

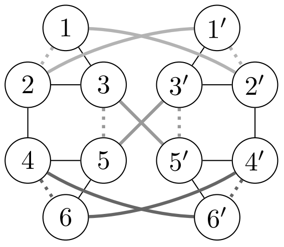

Figure 9: Graph G', which is a 2-product of G from Figure 8, with diameter 3. The six dotted edges were changed as indicated by the thick edges.

is 0, node v' must output 1.

Diameter and approximating it. Approximating the diameter to an arbitrary factor of α (including 1) is not CF-$hard_{\mathrm{RW}}$. To see this, observe that the graph G in Figure 8 is a factor of G' in Figure 9. Since both have diameter 3, the answers supplied by an oracle in G are also valid for corresponding nodes in G'. However, a valid output for LEADER-ELECTION in G cannot be naturally extended to obtain a valid output in G'. In contrast to that, the approximation problem α-DIAMETER-APX is RW-$hard_{\mathrm{WO}}$ for arbitrary approximation guarantees α. This is the case because a WO-algorithm can solve OR by broadcasting all input values for D rounds if the answer supplied by the oracle to α-DIAMETER-APX is D. If after D rounds of broadcasting node v received a message containing input 1, then v returns 1, otherwise v returns 0. It follows that any upper bound on the network diameter (or its size) is RW-$hard_{\mathrm{WO}}$. This is true even if not all nodes are equipped with the same upper bound, which simplifies the proof regarding the hardness result of MIN-CUT-PARTITION for RW with respect to WO in the following section.

Min-Cut. Neither MIN-CUT-VALUE nor MIN-CUT-PARTITION are CF-$hard_{\mathrm{RW}}$, because the two graphs G and G' from Figures 5 and 6 have the same minimum cut value (share the "same" partition inducing the minimum cut). Thus the same answer supplied by the oracle to G is also valid in G' for MIN-CUT-VALUE or MIN-CUT-PARTITION, respectively, but G' has twice the number of nodes than G. MIN-CUT-VALUE is also not RW-$hard_{\mathrm{WO}}$. To see that, observe that all cycles share a minimum cut size of 2. With the same argument as for the FACTOR-MULTIPLICITY problem, a WO algorithm cannot use this information to solve the OR problem. On the other hand, we find that MIN-CUT-PARTITION is RW-$hard_{\mathrm{WO}}$.

Proof (that MIN-CUT-PARTITION is RW-$hard_{\mathrm{WO}}$). We show that using a WO-algorithm $\mathcal{A}$ with access to an (apply-once) oracle for the MIN-CUT-PARTITION problem, every node v can determine an upper bound on the diameter. Since α-DIAMETER-APX is RW-$hard_{\mathrm{WO}}$ for any α, this is sufficient to prove our claim. Given a *partition* of the network into *black* and *white* nodes, we refer to a node which has a neighbor in the opposite partition as a *border node*. By the term *depth* of a node v we denote the minimum distance of v to a border node within the same partition, i.e, border nodes have depth zero. Observing that the degree of each node is an upper bound on the cut size, the main idea is now to bound the diameter of the network in terms of the maximum depth of a node in each partition.

To accomplish that, algorithm $\mathcal{A}$ proceeds in three stages: The only purpose of the first stage is to locally gather necessary information from the oracle prior to the second stage. The main stage of $\mathcal{A}$ is the second one, in which nodes compute an upper bound on the diameter of each partition individually. Lastly, the third stage's role is to combine the two individual upper bounds to compute an upper bound on the diameter of the whole network, and disseminate the bound throughout the network.

In the beginning of the first stage of algorithm $\mathcal{A}$ each node v invokes the MIN-CUT-PARTITION-oracle to determine whether it is in the black or in the white partition. Thereafter, node v sends a message containing the name of its partition to all of its neighbors, so that every node can determine whether it is a border

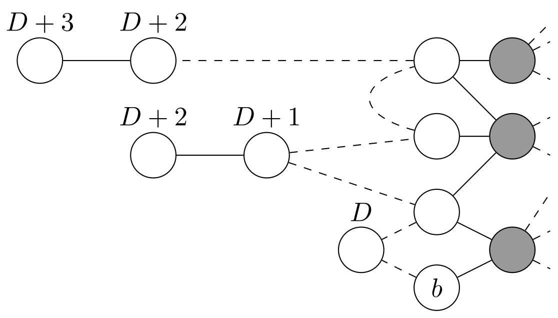

Figure 10: Illustration of the proof that MIN-CUT-PARTITION is RW-$hard_{\mathrm{WO}}$. A border node b that has observed a node at depth $D, D+1$ or $D+2$ will wait long enough to receive a message from the closest node next in depth.

node. Every non-border node v initializes its depth value $d(v) \leftarrow \infty$, while border nodes b set $d(b) \leftarrow 0$. Additionally, border nodes b initialize a value $D(b) = 0$ to keep track of the maximum observed depth inside b's partition. Following that, all nodes enter the second stage of $\mathcal{A}$.

The first round of the second stage starts with all border nodes b sending the message $(D \geq 0)$ to all neighbors within the same partition. This initiates parallel breadth-first searches inside each partition to determine the depth of every node, and to report back the maximum depth of a node inside each partition. More specifically, when a node v with $d(v) < \infty$ receives a message $(D \geq i)$ for some i, then v forwards this message to all members within the same partition. If on the other hand a node v with $d(v) = \infty$ receives a message $(D \geq i)$, then v does not forward this message, but instead it sets $d(v) \leftarrow i+1$ and broadcasts the message $(D \geq i+1)$ among all members of the same partition. A border node b that receives a message $(D \geq i)$ additionally updates $D(b) \leftarrow i$ accordingly, thus keeping track of the maximum observed depth of a node inside its partition. The crucial point is that a border node b enters stage three, if it does not receive an update to $D(b)$ in round $2D(b) \cdot (\deg(b)+1) + 2$ of stage two. In other words, a border node b adjusts the time at which it will enter stage three of algorithm $\mathcal{A}$ with each update to $D(b)$.

When both nodes b and w of a cut edge $\{b, w\}$ have entered stage three of algorithm $\mathcal{A}$, both b and w exchange their corresponding values $D(w)$ and $D(b)$. If a border node b has exchanged $D(b)$ with its neighbor w, broadcasts the message $(\mathrm{diam} \leq (2 \cdot \deg(b)+1) \cdot (D(b) + D(w)))$. Nodes v receiving a message $(\mathrm{diam} \leq i)$ for the first time enter stage three and set their output register to i after forwarding the message to all their neighbors.

We have to prove two things, namely that the upper bound broadcast by a border node is indeed an upper bound for the diameter, and that no border node broadcasts an incorrect value prematurely. To see the latter, it suffices to show that any border node b as illustrated in Figure 10 will wait long enough before starting with the third stage of the algorithm. Our proof will be by induction on the maximum depth $d(v)$ of a node v, where we show that if b has received a message $(D \geq D(b))$, then it will also receive a message indicating a maximum depth of $D(b)+1$ from a node of that depth that is closest to b, if there is one. The initial step is for $d(v) = 0$, i.e., the partition of b contains only border nodes, in which case there is nothing to show. For the induction step, let v be a node with depth $d(v)$, and assume the border node b has learned the maximum depth of a node within the partition is at least $D(b) = d(v) - 1$ by the induction hypothesis. Let b' denote a border node closest to b having a node v' at distance $d(v)$ with $d(v') = \mathit{depth}(v)$. The distance from b to b' is bounded by $2 \cdot \deg(b) \cdot D(b)$, since the shortest path between b and b' that remains inside the same partition contains at most $\deg(b)$ border nodes, and the distance between any two border nodes on that path is at most $2 \cdot D(b)$. Any node u with depth i will broadcast its distance in round i of stage two, and this message will arrive at the closest border nodes (in distance i) after $2i$ rounds. Thus, node b' will start forwarding the message $(D \geq d(v'))$ received from v' in round $t_{v',b'} = 2d(v')$ of stage two, and this message has to travel at most additional $t_{b',b} = 2 \cdot \deg(b) \cdot D(b)$ steps to reach b. But $t_{v,b'} + t_{b',b} = 2 \cdot (D(b)+1) + 2 \cdot \deg(b) \cdot D(b)$ is exactly the round until which b waits for a message indicating a possibly larger depth. Thus, no border node prematurely transmits an incorrect partition depth to its neighboring nodes in the opposite partition.

Lastly, the found value is indeed an upper bound on the diameter by a similar argument. The degree $\deg(b)$ of b in is an upper bound for the size of the cut, and therefore a shortest path between any two nodes u and v can cross the cut at most $\deg(b)$ times and contains at most $\deg(b)$ border nodes. $\square$

It follows that MIN-CUT is also RW-*hard*$_{\text{WO}}$, since the output includes the value of a minimum cut and, in particular, a valid output to MIN-CUT-PARTITION. In fact, MIN-CUT is CF-*hard*$_{\text{RW}}$ as is established in the following proof.

Proof (that MIN-CUT is CF-*hard*$_{\text{RW}}$)**.** We describe an algorithm $\mathcal{A}$ that elects a leader among all border nodes of the white partition. This will be accomplished by choosing a new random identifier for every white border node in each round. The main insight is that it can be checked whether every node tossed a unique identifier by counting the number of cut edges incident to different identifiers, and comparing this number to the size of the cut provided by an oracle. To that end, in a preliminary step of $\mathcal{A}$ every node v once invokes the oracle to MIN-CUT so that v is supplied with the value of the minimum cut k and its partition denoted by *black* or *white*. After obtaining an answer from the oracle every node sends a message indicating the partition it is in to all neighbors, enabling every node v to determine the number $c(v)$ of cut edges incident to v.

For the remainder of algorithm $\mathcal{A}$ all white border nodes w, i.e., nodes inside the white partition with $c(w) > 0$, are referred to as *candidate leaders*. Nodes inside the black partition and white nodes with $c(v) = 0$ set their output register $\rho \leftarrow$ "not leader". In every round r, every candidate leader w chooses an identifier $\beta_r(w)$ by appending one random bit to the previous identifier $\beta_{r-1}(w)$, and broadcasts the message $M = (r, \beta_r(w), c(w))$ among all white nodes. If some node v receives two messages M, M' containing the same round numbers r and identifiers $\beta_r(w)$, but a different number of cut edges $c(w)$, then v broadcasts an *inhibiting message* for round r. For all candidate leaders w denote by $M_w(s)$ the set of different messages M that were sent in round s and received by w so far. Denote further by $c_w(s) := \sum_{(s,\beta,c)\in M_w(s)} c$ the total number of cut edges from different nodes received by w for round s, and by s_w the smallest non-inhibited round s for w that satisfies $c_w(s) = k$, i.e., the first non-inhibited round in which every candidate leader tossed a different identifier. Whenever s_w is not defined in round r, node w sets its output register $\rho \leftarrow \varepsilon$. When on the other hand s_w is defined in round r, node w checks whether the identifier it chose in round s_w was the smallest among those appearing in $M_w(s_w)$. If this is the case, node w sets $\rho \leftarrow$ "leader", otherwise it sets $\rho \leftarrow$ "not leader".

To see that the algorithm is correct, assume for the sake of contradiction that in round r all nodes are ready and two different candidate leaders u, w output "leader". Since all nodes are ready no node is currently sending an inhibiting message, and because both u and w output "leader" the round numbers s_u and s_w are both defined and not inhibited for any node; assume w.l.o.g. that $s_u \leq s_w$. On the other hand, the value $c_u(s_u)$ must be the same as $c_w(s_w)$, namely they must both be k. This can only be the case if both u and w have received a message from all candidate leaders for round s_u and s_w respectively . In round r node w received all messages sent by other candidate leaders in round s_w, and therefore w must also have received all the messages sent by candidate leaders in round s_u. Because round s_u is not inhibited for any node we conclude that $s_u = s_w$ and since no inhibiting message is being sent in round r also $M_u(s_u) = M_w(s_w)$ must hold, contradicting that both u and w output "leader". Lastly, algorithm $\mathcal{A}$ will reach a ready configuration after every candidate leader tossed a unique identifier. $\square$

One can also give a WO-algorithm solving LEADER-ELECTION with access to a MIN-CUT oracle by using a more careful construction and analysis. It is however easier to see that MIN-CUT is CF-*hard*$_{\text{WO}}$ by applying Theorem 7, i.e., because MIN-CUT is both CF-*hard*$_{\text{RW}}$ and RW-*hard*$_{\text{WO}}$ it must also be CF-*hard*$_{\text{WO}}$. This completes our effort to identify in which classes each of the presented problems lie, and we turn ourselves to proving the missing link in the last argument, namely Theorem 7.

5 Proof of Theorem 7

The techniques introduced in Section 2 together with the completeness result for Or found in Section 4.2.1 allow us to present a proof for Theorem 7. A key ingredient in the proof is the notion of a *fork*, which is a sub-process of the execution dedicated to simulating some algorithm $\mathcal{A}$. The fork's name $[r]$ will indicate the round number in which the simulation was started. A fork $[r]$ dedicated to $\mathcal{A}$ encapsulates the complete state required to simulate $\mathcal{A}$, and messages sent and received by $[r]$ are identified by the fork's name.

The theorem states that if a problem Π is both CF-$hard_{\text{RW}}$ and RW-$hard_{\text{WO}}$, then it is also CF-$hard_{\text{WO}}$. Let $\Pi \in$ CF-$hard_{\text{RW}} \cap$ RW-$hard_{\text{WO}}$ be a problem satisfying the premise. Denote by $\mathcal{A}_{\text{LE}}$ a RW-algorithm solving Leader-Election with an access to a Π-oracle, and by $\mathcal{A}_{\text{OR}}$ a WO-algorithm solving Or with an access to a Π-oracle respectively. Employing Lemma 4, we assume that $\mathcal{A}_{\text{LE}}$ in fact sustainably solves Leader-Election. We wish to establish the assertion by presenting a WO-algorithm $\mathcal{A}$ solving Leader-Election with access to a Π-oracle.

Of course, algorithm $\mathcal{A}$ cannot directly simulate $\mathcal{A}_{\text{LE}}$ because it is a RW-algorithm. We would therefore like to perform multiple simulations of $\mathcal{A}_{\text{OR}}$ in order to detect a ready configuration of $\mathcal{A}_{\text{LE}}$. Unfortunately, these multiple simulations cannot be carried out concurrently since each one of them requires its own independent access to the Π-oracle, whereas $\mathcal{A}$ accesses the Π-oracle only once per round. Instead, we will use a careful forking mechanism to schedule disjoint accesses to this scarce resource.

Algorithm $\mathcal{A}$ simulates $\mathcal{A}_{\text{LE}}$ in phases, starting from phase 1, where each phase p is responsible for executing round p of the simulation of $\mathcal{A}_{\text{LE}}$. Indeed, in round 1 of phase p, node v executes round p of this simulation accessing the Π-oracle. Following that, node v initiates a fork called $[p]$ dedicated to the simulation of $\mathcal{A}_{\text{OR}}$. The input to fork $[p]$ is 0 if v was ready in round p under $\mathcal{A}_{\text{LE}}$ (v observes that from the outcome of round 1 of phase p); the input is 1 otherwise. In the next p rounds of the phase, forks $[1], [2], \ldots, [p]$ (all dedicated to $\mathcal{A}_{\text{OR}}$) are executed, one fork per round (say, in lexicographic order), so in total phase p consists of $p + 1$ rounds. The output of $\mathcal{A}$ is determined as follows: if fork $[r]$ for some $r \leq p$ has output 0 during phase p, then v writes the output value of $\mathcal{A}_{\text{LE}}$'s round r (which was obtained during phase r) to $\mathcal{A}$'s output register.

The fixed execution order of the forks simulating $\mathcal{A}_{\text{OR}}$ guarantees that every fork $[p]$ is executed in a synchronized manner, that is, all nodes execute round r of this fork in the same round under $\mathcal{A}$. The logic of Or guarantees that fork $[r]$ of $\mathcal{A}_{\text{OR}}$ has output 0 if and only if round r under $\mathcal{A}_{\text{LE}}$'s simulation is in a ready configuration. Since $\mathcal{A}_{\text{OR}}$ is a WO-algorithm, node v can immediately rely on a returned 0 value to conclude that this indeed happened. Moreover, as $\mathcal{A}_{\text{LE}}$ is sustainably solving the leader election problem, the output returned by v under $\mathcal{A}$ must lead to a correct output for Leader-Election, thus establishing Theorem 7.

References

[1] Abrahamson, K., Adler, A., Higham, L., Kirkpatrick, D.: Probabilistic solitude verification on a ring. In: Proceedings of the fifth annual ACM symposium on Principles of distributed computing (1986)

[2] Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. Journal of Algorithms 7(4), 567 – 583 (1986)

[3] Angluin, D.: Local and global properties in networks of processors (extended abstract). In: Proceedings of the twelfth annual ACM symposium on Theory of computing (1980)

[4] Angluin, D., Aspnes, J., Chan, M., Fischer, M., Jiang, H., Peralta, R.: Stably computable properties of network graphs. In: Prasanna, V., Iyengar, S., Spirakis, P., Welsh, M. (eds.) Distributed Computing in Sensor Systems (2005)

[5] Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing (2004)

[6] Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing (2006)

[7] Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing 20, 279–304 (2007)

[8] Angluin, D., Fischer, M., Jiang, H.: Stabilizing consensus in mobile networks. In: Gibbons, P., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) Distributed Computing in Sensor Systems (2006)

[9] Aspnes, J., Ruppert, E.: An introduction to population protocols. In: Garbinato, B., Miranda, H., Rodrigues, L. (eds.) Middleware for Network Eccentric and Mobile Applications (2009)

[10] Boldi, P., Vigna, S.: Computing anonymously with arbitrary knowledge. In: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing (1999)

[11] Boldi, P., Vigna, S.: An effective characterization of computability in anonymous networks. In: Proceedings of the 15th International Conference on Distributed Computing (2001)

[12] Boldi, P., Vigna, S.: Universal dynamic synchronous self–stabilization. Distributed Computing 15(3), 137–153 (2002)

[13] Chalopin, J., Das, S., Santoro, N.: Groupings and pairings in anonymous networks. In: Proceedings of the 20th international conference on Distributed Computing (2006)

[14] Chalopin, J., Godard, E., Métivier, Y.: Local terminations and distributed computability in anonymous networks. In: Taubenfeld, G. (ed.) Distributed Computing (2008)

[15] Dolev, S.: Self-Stabilization (2000)

[16] Dolev, S., Israeli, A., Moran, S.: Uniform dynamic self-stabilizing leader election. In: Toueg, S., Spirakis, P., Kirousis, L. (eds.) Distributed Algorithms (1992)

[17] Emek, Y., Wattenhofer, R.: Stone age distributed computing. In: PODC (2013)

[18] Flocchini, P., Kranakis, E., Krizanc, D., Luccio, F.L., Santoro, N.: Sorting and election in anonymous asynchronous rings. J. Parallel Distrib. Comput. 64(2), 254–265 (Feb 2004)

[19] Fraigniaud, P., Korman, A., Peleg, D.: Local distributed decision. In: Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on (oct 2011)

[20] Fraigniaud, P., Halldórsson, M., Korman, A.: On the impact of identifiers on local decision. In: Baldoni, R., Flocchini, P., Binoy, R. (eds.) Principles of Distributed Systems (2012)

[21] Fraigniaud, P., Ilcinkas, D., Pelc, A.: Oracle size: A new measure of difficulty for communication tasks. In: Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing (2006)

[22] Fraigniaud, P., Korman, A., Parter, M., Peleg, D.: Randomized distributed decision. In: DISC (2012)

[23] Fraigniaud, P., Rajsbaum, S., Travers, C.: Locality and checkability in wait-free computing. In: Peleg, D. (ed.) DISC (2011)

[24] Godsil, C.D., Royle, G.: Algebraic Graph Theory (2001)

[25] Göös, M., Suomela, J.: Locally checkable proofs. In: Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing (2011)

[26] Guerraoui, R., Ruppert, E.: What can be implemented anonymously? In: Proceedings of the 19th

international conference on Distributed Computing (2005)
[27] Herlihy, M., Rajsbaum, S.: A classification of wait-free loop agreement tasks. Theoretical Computer Science 291(1), 55 – 77 (2003), <ce:title>Distributed Computing</ce:title>
[28] Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM 46(6), 858–923 (Nov 1999)
[29] Israeli, A., Itai, A.: A fast and simple randomized parallel algorithm for maximal matching. Information Processing Letters 22(2), 77 – 80 (1986)
[30] Itai, A., Rodeh, M.: Symmetry breaking in distributive networks. In: Proc. 22nd Annual Symp. Foundations of Computer Science SFCS '81 (1981)
[31] Korman, A., Kutten, S., Peleg, D.: Proof labeling schemes. In: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing (2005)
[32] Korman, A., Sereni, J.S., Viennot, L.: Toward more localized local algorithms: removing assumptions concerning global knowledge. In: Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing (2011)
[33] Linial, N.: Locality in distributed graph algorithms. SIAM Journal on Computing 21(1), 193–201 (1992)
[34] Liu, X., Xu, Z., Pan, J.: Classifying rendezvous tasks of arbitrary dimension. Theoretical Computer Science 410(21–23), 2162 – 2173 (2009)
[35] Luby, M.: A simple parallel algorithm for the maximal independent set problem. In: Proceedings of the seventeenth annual ACM symposium on Theory of computing (1985)
[36] Lynch, N.A.: Distributed Algorithms (1996)
[37] Mavronicolas, M., Michael, L., Spirakis, P.: Computing on a partially eponymous ring. In: Proceedings of the 10th international conference on Principles of Distributed Systems (2006)
[38] Métivier, Y., Robson, J.M., Zemmari, A.: Analysis of fully distributed splitting and naming probabilistic procedures and applications. In: Moscibroda, T., Rescigno, A. (eds.) SIROCCO (2013)
[39] Naor, M., Stockmeyer, L.: What can be computed locally? SIAM Journal on Computing 24(6), 1259–1277 (1995)
[40] Schieber, B., Snir, M.: Calling names on nameless networks. In: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing (1989)
[41] Suomela, J.: Survey of local algorithms. ACM Comput. Surv. (to appear), preliminary version
[42] Yamashita, M., Kameda, T.: Computing on anonymous networks: Part i-characterizing the solvable cases. IEEE Trans. Parallel Distrib. Syst. 7(1), 69–89 (Jan 1996)