Cheating by Duplication: Equilibrium Requires Global Knowledge

Yehuda Afek, Shaked Rafaeli, and Moshe Sulamy

Tel-Aviv University

Abstract

Distributed algorithms with rational agents have always assumed the size of the network is known to the participants before the algorithm starts. Here we address the following question: what global information must agents know a-priori about the network in order for equilibrium to be possible? We start this investigation by considering different distributed computing problems and showing how much each agent must a-priori know about n, the number of agents in the network, in order for distributed algorithms to be equilibria.

We prove that when n is not a-priori known, equilibrium for both knowledge sharing and coloring is impossible. We provide new algorithms for both problems when n is a-priori known to all agents. We further show that when agents are given a range in which the actual value of n may be, different distributed problems require different such ranges in order for equilibrium to be possible. By providing algorithms that are equilibrium on the one hand and impossibility results on the other, we provide the tight range in which equilibrium is possible but beyond which there exist no equilibrium for the following common distributed problems: Leader Election, Knowledge Sharing, Coloring, Partition and Orientation.

*This research was supported by the Israel Science Foundation (grant 1386/11).

1 Introduction

The complexity and simplicity of most distributed computing problems depend on the inherent a-priori knowledge given to all participants. Usually, the more information processors in a network start with, the more efficient and simple the algorithm for a problem is. Sometimes, this information renders an otherwise unsolvable problem, solvable.

Our model is a distributed setting where *rational* agents [3] participate in the algorithm and may deviate from it when a deviation is deemed more profitable for them, i.e., the output they want becomes more likely. For example, in a distributed online game a participant may want (or may not want) to host the game server, i.e., to be the leader (or not) in an election algorithm. Another example is a distributed frequency assignment in which an agent prefers a certain frequency (color) for which he already has an antenna.

To differentiate from Byzantine faults, we require the *Solution Preference* property that ensures agents never prefer an outcome in which the algorithm fails over an outcome with a legal output. Previous works in this setting [3, 4, 17] assumed agents a priori know n, the number of agents in the network (henceforth called *actual number*).

In this paper we examine the a-priori knowledge about n required for equilibrium in a distributed network of *rational* agents, each of which has a preference over the output. Notice that when n is not a-priori known, agents may *duplicate* themselves to affect the outcome of the algorithm. We examine several problems which in standard distributed computing without rational agents can be solved without knowledge about n. But in a setting of rational agents the a-priori knowledge about n plays a more crucial role. For example, in the $LOCAL$ [21] model, coloring has a radius of $log*(n)$, but with rational agents and with no knowledge about n, equilibrium is impossible.

Intuitively, the more an agent duplicates itself, the greater its power to affect the output of the algorithm becomes. For every problem, we strive to find the maximum number of duplications in which the duplicating agent cannot affect the output, i.e., an equilibrium is still possible. This maximum number of duplications also depends on whether other agents can detect that a duplication has taken place, by detecting that the network could not possibly be this large. To detect this situation they need to possess knowledge about the network size, or of what is an impossibly large network.

In this paper, we translate this intuition into a precise computation of the relation between the lower bound α and the upper bound $\beta = f(\alpha)$ on n, that must be a-priori known in order for equilibrium to be possible. We denote this relation f-bound.

To find out the f-bound of a problem, we first figure out what is the minimum number of duplications for which equilibrium is impossible when n is not known at all, and then show an algorithm that is an equilibrium when the amount of duplications is limited to that number, as well as when n itself is a-priori known. Finally, we calculate the f-bound by taking into account both an agent's possible gain by duplicating itself and the risk it takes of being caught.

These are the main contributions of the current paper:

- Theorem 3.3: Coloring has no equilibrium when agents have no a-priori knowledge about n.
- Section 4.2: A Coloring algorithm that is an equilibrium when agents a-priori know n.
- Theorem 3.1: Knowledge Sharing (see [4] and our model) has no equilibrium when agents have no a-priori knowledge about n.
- Theorem 4.1: A Knowledge Sharing algorithm that is an equilibrium when a cheating agent pretends to be at most n agents.

- Theorem 5.3: Knowledge Sharing is $(2\alpha-2)$-bound in rings, i.e., there is a Knowledge Sharing algorithm that is an equilibrium when the upper and lower bounds on n satisfy $\alpha \leq \beta \leq 2\alpha-2$, and Knowledge Sharing has no equilibrium when $\beta > 2\alpha - 2$.
- Theorem 5.5 and Corollary5.4: Coloring and 2-Knowledge Sharing are ∞-bound in rings, i.e., there is an equilibrium given *any* finite bound, but no equilibrium exists if no bound or information about n is a-priori given.
- Theorem 5.6: Leader Election is $(\alpha + 1)$-bound.
- Theorems 5.7 and 5.8: Ring Partition and Orientation are unbounded, i.e., there is an equilibrium even when neither n nor any bound on n is given.

1.1 Related Work

The connection between distributed computing and game theory stemmed from the problem of secret sharing [27]. Further works continued the research on secret sharing and multiparty computation when both Byzantine and rational agents are present [1, 11, 13, 15, 23, 16].

Another line of research presented the BAR model (Byzantine, acquiescent [32] and rational) [5, 25, 32], while a related line of research discusses converting solutions with a mediator to cheap talk [1, 2, 8, 9, 12, 19, 24, 28, 30, 31].

Abraham, Dolev, and Halpern [3] were the first to present protocols where processors in the network behave as rational agents, specifically protocols for Leader Election. In [4] the authors continue this line of research by providing basic building blocks for game theoretic distributed algorithms, namely a wake-up a knowledge sharing equilibrium building block. Algorithms for consensus, renaming, and leader election are presented using these building blocks. Consensus was researched further by Halpern and Vilaca [17], who showed that there is no ex-post Nash equilibrium, and a Nash equilibrium that tolerates f failures under some minimal assumptions on the failure pattern.

Coloring and Knowledge Sharing have been studied extensively in a distributed setting [6, 7, 10, 18, 20, 21, 29]. An algorithm for Knowledge Sharing with rational agents was presented in [4], while Coloring with rational agents has not been studied previously, to the best of our knowledge.

2 Model

We use the standard message-passing model, where the network is a bidirectional graph $G = (V, E)$ with n nodes, each node representing an agent, and $|E|$ edges over which they communicate. G is assumed to be 2-vertex-connected[1]. Throughout the entire paper, n always denotes the actual number of nodes in the network.

In our model the computational power of each processor is unbounded, instantaneous and can be performed between rounds. Given the necessary information over its communication links, an agent can perform any calculation or simulation, and can complete it by the time it sends the next communication message.

Initially, each agent knows its *id* and input, but not the *id* or input of any other agent. We assume the prior of each agent over any information it does not know is uniformly distributed over all possible values. Each agent is assigned a unique *id*, taken from the set of natural numbers.

[1] This property was shown necessary in [4], since if such a node exists it can alter any message passing through it. Such a deviation cannot be detected since all messages between the sub-graphs this node connects must traverse through it. This node can then skew the algorithm according to its preferences.

Furthermore, we assume all agents start the protocol together, i.e., all agents wake-up at the same time. If not, we can use the Wake-Up [4] building block to relax this assumption.

2.1 Equilibrium in Distributed Algorithms

Informally, a distributed algorithm is an equilibrium if no agent at no point in the execution can do better by unilaterally deviating from the algorithm. When considering a deviation, an agent assumes all other agents follow the algorithm, even if they can do better by not following it. We assume algorithms always abort whenever a deviation is detected by another agent, even if the detecting agent can gain by not aborting. In this manner, our notion of equilibrium is basically Bayes-Nash equilibrium, but not subgame perfect equilibrium [26].

Each node in the network is a *rational* agent, following the model in [4]. The algorithms produce a single output per agent, once, at the end of the execution. In this paper, each agent has a preference only over its own output.

Formally, let o_a be the output of agent a, let Θ be the set of all possible output vectors, and denote the output vector $O = \{o_1, \ldots, o_n\} \in \Theta$, where $O[a] = o_a$. Let Θ_L be the set of *legal* output vectors, in which the protocol terminates successfully, and let Θ_E be the set of *erroneous* output vectors, such that $\Theta = \Theta_L \cup \Theta_E$ and $\Theta_L \cap \Theta_E = \varnothing$.

Each agent a has a utility function $u_a : \Theta \to \mathbb{N}$. The higher the value assigned by u_a to an output vector, the better this vector is for a. To differentiate rational agents from Byzantine faults, we assume the utility function satisfies *Solution Preference* [3, 4] which guarantees that an agent never has an incentive to cause the algorithm to fail.

Definition 2.1 (Solution Preference)**.** The utility function u_a of an agent a never assigns a higher utility to an erroneous output than to a legal one, i.e.:

$$\forall a, O_L \in \Theta_L, O_E \in \Theta_E : u_a(O_L) \geq u_a(O_E)$$

We differentiate the *legal* output vectors, which ensure the output is valid and not erroneous, from the *correct* output vectors, which are output vectors that are a result of a correct execution of the algorithm, i.e., without any deviation. The Solution Preference guarantees agents never prefer an erroneous output. However, they may prefer a *legal* but *incorrect* output.

Recall that we assume agents only have preferences over their own output, i.e., for any $O_1, O_2 \in \Theta_L$ where $O_1[a] = O_2[a]$, $u_a(O_1) = u_a(O_2)$. For simplicity, we also assume each agent a has a *single* preferred output value p_a, and we normalize the utility function values, such that[2]:

$$u_a(O) = \begin{cases} 1 & o_a = p_a \text{ and } O \in \Theta_L \\ 0 & o_a \neq p_a \text{ or } O \in \Theta_E \end{cases} \tag{1}$$

Our results hold for *any* utility function that satisfies Solution Preference.

Definition 2.2 (Expected Utility)**.** Let r be a round in a specific execution of an algorithm. Let a be an arbitrary agent. For each possible output vector O, let $x_O(s, r)$ be the probability, estimated by agent a at round r, that O is output by the algorithm if a takes step s [3], and all other agents follow the algorithm. The *Expected Utility* a estimates for step s in round r of that specific execution is:

$$\mathbb{E}_{s,r}[u_a] = \sum_{O \in \Theta} x_O(s, r) u_a(O)$$

[2] This is the weakest assumption that satisfies Solution Preference, since it gives cheating agents the highest incentive to deviate. A utility assigning a lower value for failure than $o_a \neq p_a$ would deter a cheating agent from deviating.

Note that agents can also estimate the expected utility of *other* agents by simply considering a different utility function.

An agent will deviate whenever the deviating step leads to a strictly higher expected utility than the expected utility of the next step of the algorithm. By the utility function 1, an agent will prefer *any* deviating step that increases the probability of getting its preferred output, even if that deviating step also increases the risk of an erroneous output.

Formally, an agent a can simulate (locally) all possible executions leading to its current state, and for each execution calculate every possible continuation and its output. For each possible continuation and corresponding output, Agent a can calculate the probability x_O based on its prior (which is uniformly distributed) and the communication it had received so far, assuming other agents follow the algorithm. For a deterministic step in the continuation, there is a single continuation. For a non-deterministic step, Agent a calculates the possible continuations according to the probability distributions dictated by the algorithm, or a uniform distribution if the the algorithm does not dictate, over what is unknown to a.

Let Λ be an algorithm. If by deviating from Λ and taking step s, the expected utility of a is higher, we say that agent a has an *incentive to deviate* (i.e., cheat). For example, at round r algorithm Λ may dictate that a flips a fair binary coin and sends the result to all of its neighbors. Any other action by a is considered a *deviation*: whether the message was not sent to all neighbors, sent later than it should have, or whether the coin toss was not fair, e.g., a only sends 0 instead of a random value. If no agent can unilaterally increase its expected utility by deviating from Λ, we say that the protocol is an *equilibrium*. We assume a *single* deviating agent, i.e., there are no coalitions of agents.

Definition 2.3 (Distributed Equilibrium)**.** Let $s(r)$ denote the next step of algorithm Λ in round r. Λ is an equilibrium if for any deviating step s, at any round r of every possible execution of Λ:

$$\forall a, r, s : \mathbb{E}_{s(r),r}[u_a] \geq \mathbb{E}_{s,r}[u_a]$$

2.2 Knowledge Sharing

The Knowledge Sharing problem (adapted from [4]) is defined as follows:

1. Each agent a has a private input i_a, in addition to its *id*, and a function q, where q is identical at all agents.

2. A Knowledge Sharing protocol terminates *legally* if all agents output the *same* value, i.e., $\forall a, b : o_a = o_b \neq \perp$. Thus the set Θ_L is defined as: $O \in \Theta_L \iff \forall a, b : O(a) = O(b) \neq \perp$.

3. A Knowledge Sharing protocol terminates *correctly* (as described in Section 2.1) if each agent outputs at the end the value $q(I)$ over the input values $I = \{i_1, \ldots, i_n\}$ of all other agents[4].

4. The function q satisfies the Full Knowledge property:

   Definition 2.4 (Full Knowledge Property)**.** A function q fulfills the *full knowledge* property if, for each agent that does not know the input values of all other agents, any output in the range of q is *equally* possible. Formally, for any $1 \leq j \leq m$, fix $(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_m)$

[3] A step specifies the entire operation of the agent in a round. This may include drawing a random number, performing any internal computation, and the contents and timing of any message delivery.

[4] Notice that any output is legal as long as it is the output of all agents, but only a single output value is considered *correct* for a given input vector.

and denote $z_y = |\{x_j | q(x_1, \ldots, x_j, \ldots, x_m) = y\}|$. A function q fulfills the *full knowledge* property if, for any possible output y in the range of q, z_y is the same[5].

We assume that each agent a prefers a certain output value p_a.

2-Knowledge Sharing The 2-Knowledge Sharing problem is a Knowledge Sharing problem with exactly 2 distinct possible output values.

2.3 Coloring

In the Coloring problem [10, 20], the output o_a of each agent a is a color. Θ_L is any O such that $\forall a : o_a \neq \perp$ and $\forall (a, b) \in E : o_a \neq o_b$.

We assume that every agent a prefers a specific color p_a.

3 Impossibility With No Knowledge

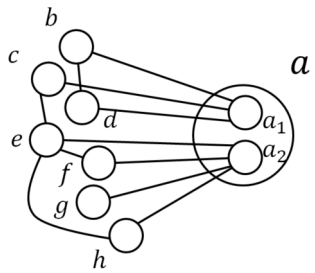
Here we show that without any a-priori knowledge about n, there is no algorithm that is an equilibrium for both Knowledge Sharing and Coloring.

A fundamental building block in many algorithms is *Wake-Up* [4], in which agents learn the graph topology. If n is not known at all, how can an agent be sure the topology it learned is correct?

Let a be a malicious agent with δ outgoing edges. A possible deviation for a is to simulate imaginary agents a_1, a_2 and to answer over some of its edges as a_1, and over the others as a_2, as illustrated in Figure 1. From this point on a acts as if it is 2 agents. Here we assume that the *id* space is much larger than n, allowing us to disregard the probability that the fake *id* collides with an existing *id*.

Note that an agent may be forced by the protocol to commit to its fake duplication early, by starting the algorithm with a process that tries to map the graph topology, such as the Wake-Up [4] algorithm. If an algorithm does not begin by mapping the topology, an agent could begin the protocol as a single agent and duplicate itself at a later stage. This allows the agent to "collect information" and increase its ability to effect the output. Since a Wake-Up protocol can be added at the beginning of every algorithm, we assume every algorithm starts by mapping the network, thus forcing a duplicating agent to commit to its duplication scheme at the beginning of the algorithm.

Figure 1: Agent a acting as separate agents a_1, a_2



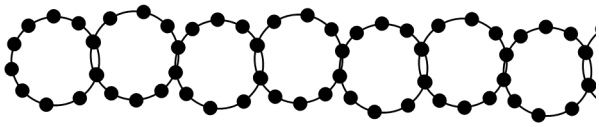
Regarding the output vector, notice that an agent that pretends to be more than one agent still outputs a *single* output at the end. The duplication causes agents to execute the algorithm as if it

[5]The definition assumes input values are drawn uniformly, otherwise the definition of z_y can be expanded to the sum of probabilities over every input value for x_j.

is executed on a graph G' (with the duplicated agents) instead of the original graph G; however, the output is considered legal if $O = \{o_a, o_b, \dots\} \in \Theta_L$ rather than if $\{o_{a_1}, o_{a_2}, o_b, \dots\} \in \Theta_L$.

It should be noted that even without exact knowledge of n, some types of initial knowledge are equivalent to knowing n. For example, consider a 2-connected graph where agents know that G is made out of 10-agent rings connected consecutively, but do not know the exact value n, as seen in Figure 2. In this case, no duplication by a single agent is possible; since n can be calculated correctly, a duplicating agent will be caught. In this section, we assume that no knowledge of the graph topology is a-priori known.

Figure 2: A graph of consecutive rings of a fixed size



It is important to emphasize that for any non-trivial distributed algorithm, the outcome cannot be calculated using only private data without communication, i.e., for rational agents, no agent can calculate the outcome privately at the beginning of the algorithm. This means that at round 0, for any agent a and any step s of the agent that does not necessarily result in the algorithm failure, it must hold that: $\mathbb{E}_{s,0}[u_v] \notin \{0, 1\}$ (a value of 0 means an agent will surely not get its preference, and 1 means it is guaranteed to get its preference).

In this section we label agents in graph G as $a_1, ..., a_n$, set in a clockwise manner in a ring and arbitrarily in any other topology. These labels are not known to the agents themselves.

3.1 Impossibility of Knowledge Sharing

Theorem 3.1. *There is no algorithm for Knowledge Sharing that is an equilibrium in a 2-connected graph when agents do not know n.*

Proof. Assume by contradiction that Λ is a Knowledge Sharing algorithm that is an equilibrium in any graph without knowing n. Let D, E be two 2-connected graphs of rational agents. Consider the execution of Λ on graph H created by D, E, and adding two nodes a_1, a_2 and connecting these nodes to 1 or more arbitrary nodes in both D and E (see Figure 3).

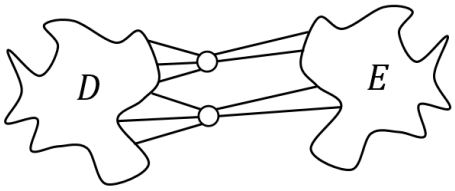


Figure 3: Graph H created by two arbitrary sub-graphs D,E

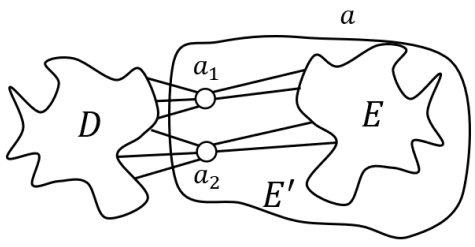


Figure 4: Example of agent a pretending to be $E' = E \cup \{a_1, a_2\}$

Recall that the vector of agents' inputs is denoted by $I = i_1, i_2, \cdots, i_n$, and $n = |H| = |D| + |E| + 2$. Let t_D be the first round after which $q(I)$ can be calculated from the collective information

that all agents in D have[6], and similarly t_E the first round after which $q(I)$ can be calculated in E. Consider the following three cases:

1. $\boldsymbol{t_E < t_D}$: $q(I)$ cannot yet be calculated in D at round t_E. Let $E' = E \cup \{a_1, a_2\}$. Since $E \subset E'$, the collective information in E' at round t_E is enough to calculate $q(I)$. Since n is not known, an agent a could emulate the behavior of E', making the agents believe the algorithm runs on H rather than D. In this case, this cheating agent knows at round t_E the value of $q(I)$ in this execution, but the collective information of agents in D is not enough to calculate $q(I)$, which means the output of agents in D still depends on messages from E', the cheater. Thus, if a learns that the output $q(I) \neq p_a$, it can send messages that may cause the agents in D to decide a value $x \neq q(I)$. In the case where $p_a = x$, agent a increases its expected utility by sending a set of messages different than that decreed by the protocol. Thus, agent a has an incentive to deviate, contradicting distributed equilibrium.

2. $\boldsymbol{t_D = t_E}$: both E and D have enough collective information to calculate $q(I)$ at the same round. The collective information in E at round t_E already exists in E' at round $t_E - 1$. Since $t_D = t_E$, the collective information in D is not enough to calculate $q(I)$ in round $t_E - 1$. Thus, similarly to Case 1, a can emulate E' and has an incentive to deviate.

3. $\boldsymbol{t_E > t_D}$: Symmetric to Case 1.

Thus, Λ does not is not an equilibrium for the Knowledge Sharing problem. □

When $|D| = |E|$, the proof of Theorem 3.1 brings us to the following corollary:

Corollary 3.2. *When a cheating agent pretends to be more than n agents, there is no algorithm for Knowledge Sharing that is an equilibrium when agents have no a-priori knowledge of n.*

3.2 Impossibility of Coloring in a Ring

The proof of Theorem 3.1 relies on the Full Knowledge property of the Knowledge Sharing problem, i.e., no agent can calculate the output before knowing all the inputs. The Coloring problem, however, is a more local problem [22], and nodes may color themselves without knowing anything about far distant nodes.

Theorem 3.3. *There is no algorithm for Coloring that is an equilibrium in a ring when agents have no a-priori knowledge of n.*

Proof. In order to show an incentive to deviate, we generalize the notion of *expected utility*. Recall that an agent outputs a *single* color, even if it pretends to be several agents. In Coloring, a cheating agent only wishes to influence the output color of its *original neighbors* to enable it to output its preferred color while maintaining the legality of the output.

Definition 3.4 (Group Expected Utility)**.** Let r be a round in an execution ε, and let M be a group of agents. For any set $S = \{s_1, \ldots, s_{|M|}\}$ of steps of agents in M, let Ψ be the set of all possible executions for which the same messages traverse the links that income and outgo to/from M as in ε until round r, and in round r each agent in M takes the corresponding step in S. For each possible output vector O, let x_O be the sum of probabilities over Ψ that O is decided by the protocol. For any agent v, the *Group Expected Utility* of u_v by M taking steps S at round r in execution ε is: $\mathbb{E}_{M,S,r}[u_v] = \sum_{O \in \Theta} x_O u_v(O)$.

[6] Regardless of the complexity of the computation.

Assume by contradiction that Γ is a Coloring algorithm that is an equilibrium in an n agent ring $\{a_1, \ldots, a_n\}$. Let G be a ring with a segment of k consecutive agents, $k \geq 3$, all of which have the same color preference p. Assume w.l.o.g., they are centered around a_n if k is odd and around a_n, a_1 if even. Let L be the group of agents $\{a_{n-1}, \ldots, a_{\lfloor \frac{n}{2} \rfloor + 1}\}$, and R the group of agents $\{a_1, \ldots, a_{\lceil \frac{n}{2} \rceil - 1}\}$. Denote $L' = L \cup \{a_{\lceil \frac{n}{2} \rceil}, a_n\}$ and $R' = V \setminus L'$ (see Figures 5, and 6).

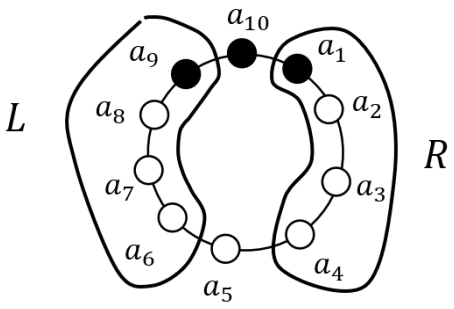


Figure 5: Ring with 3 colliding agents a_9, a_{10}, a_1 with groups L, R

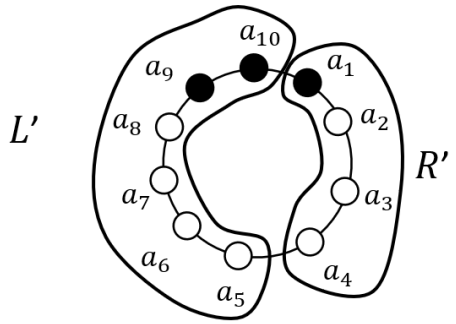


Figure 6: Ring with 3 colliding agents with groups L', R'

Definition 3.5. Let Y be a group of agents (e.g., L or R). In any round r in an execution, let $S^r(Y)$ denote the vector of steps of agents in Y according to the protocol. We say Y *knows* the utility of agent a if it holds that $\mathbb{E}_{Y,S^r(Y)}[u_a] \in \{0, 1\}$. We say Y *does not know* the utility of agent a if $0 < \mathbb{E}_{Y,S^r(Y)}[u_a] < 1$.

Recall that at round 0 no agent (or group of agents) knows its utility or the utility of any other agent. Consider an execution of Γ on ring G and the groups L, R in the following cases:

1. L does not know u_{a_n} throughout the entire execution of the algorithm, i.e., for agents in L it holds that $0 < Pr[o_n \neq p] < 1$. Then if L is emulated by a cheating agent, it has an incentive to deviate and set its output to p (as otherwise its utility is guaranteed to be 0).

2. L knows u_{a_n} at some round t_L, and R does not know u_{a_n} before round t_L. Consider round $t_L - 1$ and group L': In round t_L, L knows the utility of a_n, thus the collective information of agents in L at round t_L already exists in L' at round $t_L - 1$. If L' knows that $u_{a_n} = 1$, then it had already won; otherwise, L' knows that $u_{a_n} = 0$. Consider the group $R' \subset R$, that does not know u_{a_n} at round $t_L - 1$. If L' is emulated by a cheating agent a, it can send messages that increase its probability to output p, thus increasing its expected utility.

3. R knows u_{a_n} before round t_L: symmetric to Case 2.

Thus, Γ is not an equilibrium for the Coloring problem. □

4 Algorithms

Here we present algorithms for Knowledge Sharing (Section 4.1) and Coloring (Section 4.2). The Knowledge Sharing algorithm is an equilibrium in a ring when no cheating agent pretends to be more than n agents. The Coloring algorithm is an equilibrium in any 2-connected graph when agents a-priori know n.

4.1 Knowledge Sharing in a Ring

Here we present an algorithm for Knowledge Sharing that is an equilibrium in a ring when no cheating agent pretends to be more than n agents. Clearly, when agents a-priori know n it is an equilibrium, since a cheating agent is further constrained not to duplicate at all. At any point in the algorithm, whenever an agent recognizes that another agent has deviated from the protocol it immediately outputs $\perp$ resulting in the failure of the algorithm.

We assume a global orientation around the ring. This assumption can be easily relaxed via Leader Election [4]. Since the orientation has no effect on the output, Leader Election is an equilibrium in this application.

According to Corollary 3.2, when a cheating agent pretends to be *more* than n agents, there is no algorithm for Knowledge Sharing that is an equilibrium. On the other hand, the algorithm presented here *is* an equilibrium when the cheating agent pretends to be *no more* than n agents, proving that the bound is tight.

We start by describing the intuition behind the algorithm. Let n' be the size of the ring, which may include duplications. Since a cheater may be at most $\lceil \frac{n'}{2} \rceil$ agents, our algorithm must ensure that any group of $\lceil \frac{n'}{2} \rceil$ consecutive agents never gains enough collective information to calculate $q(I)$, the output of Knowledge Sharing, before the collective information at the rest of the ring is also enough to calculate $q(I)$.

To ensure this property, we employ a method by which agent s delivers its input i_s to some agent t at a specific round r, without revealing any information about i_s to any of the agents other than s and t. This method is used by every agent to send its input to two other agents that are distant enough to prevent a consecutive group of size $\lceil \frac{n'}{2} \rceil$ from learning this input too early.

At round n', the input values sent by this method are revealed simultaneously. Afterwards, every possible group of $\lceil \frac{n'}{2} \rceil$ consecutive agents had already committed to the inputs of all its members, so it is too late to change them. Now every agent can simply send its input around the ring.

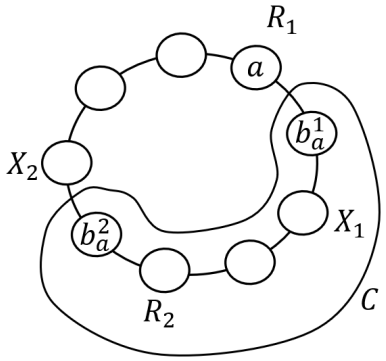


Figure 7: at round $n' - 1$: $i_a = R_1 \oplus X_1 = R_2 \oplus X_2$; group C does not know the input of a

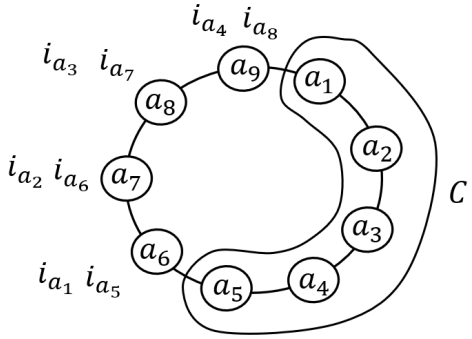


Figure 8: the input that agents $a_6, \ldots, a_9$ learn at round n'; each input of an agent in C is known by an agent not in C

Algorithm 1 describes the `Secret-Transmit` building block. The building block is called by an agent a and receives three arguments: its input i_a, a round r, and a target agent b. It assumes neighbors of b know they are neighbors of b. The building block delivers i_a at round r to agent b, and no other agent around the ring gains any information about this input. Additionally, agent b learns the input at round r and not before. In the `Secret-Transmit` building block, agent a

selects a random number R and a value X, the XOR of its input i_a with R. Each value is sent in a different direction around the ring until reaching a neighbor of b. At round $r-1$, both neighbors send the values X and R to b, thus b learns i_a at round r and no other agent around the ring has any information about i_a at round r.

Algorithm 1 Secret-Transmit(i_a, r, id_b)

1: Select a random number R
2: Let $X = R \oplus i_a$
3: Send R clockwise until it reaches a neighbor of b ▷ Each message counts down from r
Send X counter-clockwise until it reaches a neighbor of b ▷ in order to know how long to wait
4: At round $r-1$, each neighbor of b sends b the value it received, either X or R

Algorithm 2 solves Knowledge Sharing in a ring using the `Secret-Transmit` building block. All agents simultaneously transmit their input, each to 2 other agents. For each agent a, the input i_a is sent using `Secret-Transmit` to its clockwise neighbor, and to the agent that is at distance $\lfloor \frac{n'}{2} \rfloor$ counter-clockwise from a. Note that these agents form the two ends of a group C of $\lceil \frac{n'}{2} \rceil$ consecutive agents that do not include a. This guarantees that if C is a cheater pretending to be $\lceil \frac{n'}{2} \rceil \leq n$ agents, it does not learn the input i_a before round r, since at least one piece of each transmission has not reached any agent in C at any round $< r$. At round r, the agents in C already committed all of their input values to some agents in the ring that are not in C.

Algorithm 2 Knowledge Sharing in a Ring

1: All agents execute Wake-Up [4] to learn the ids of all agents and n', the size of the ring (which may include duplications)
2: For each agent a, denote b_a^1 the clockwise neighbor of a, and b_a^2 the agent at distance $\lfloor \frac{n'}{2} \rfloor$ counter-clockwise from a
3: Each agent a simultaneously performs:
`SecretTransmit`(i_a, n', b_a^1)
`SecretTransmit`(i_a, n', b_a^2)
4: At round $n'+1$, each agent sends its input around the ring
5: At round $2n'$ output $q(I)$

Theorem 4.1. *In a ring, Algorithm 2 is an equilibrium when no cheating agent pretends to be more than n agents.*

Proof. Assume by contradiction that a cheating agent pretending to be $d \leq n$ agents has an incentive to deviate from Algorithm 2, w.l.o.g., the duplicated agents are $a_1, \ldots, a_d$ (recall the indices $1, \ldots, n'$ are not known to the agents).

Let n' be the size of the ring including the duplicated agents, i.e., $n' = n + d - 1$. The clockwise neighbor of $a_{n'}$ is $a_1 = b^1_{a_{n'}}$. Denote $a_c = b^2_{a_{n'}}$ the agent at distance $\lfloor \frac{n'}{2} \rfloor$ counter-clockwise from $a_{n'}$, and note that $c \geq d$.

When $a_{n'}$ calls `Secret-Transmit` to a_1, $a_{n'}$ holds the piece R of that transmission until round $n'-1$. When $a_{n'}$ calls `Secret-Transmit` to a_c, a_{c+1} holds the piece X of that transmission until round $n'-1$. By our assumption, the cheating agent duplicated into $a_1, \ldots, a_d$. Since $d < c+1$, the cheater receives at most one piece (X or R) of each of $a_{n'}$'s transmissions before round n'. So, there is at least one input that the cheater does not learn before round n'. According to the Full Knowledge property (Definition 2.4), for the cheater at round $n'-1$ any output is equally possible,

so its expected utility for any value it sends is the same, thus it has no incentive to cheat regarding the values it sends in round $n' - 1$.

Let $a_j \in \{a_1, \ldots, a_d\}$ be an arbitrary duplicated agent. In round n', i_{a_j} is known by its clockwise neighbor $b^1_{a_j}$ and by $b^2_{a_j}$, the agent at distance $\lfloor \frac{n'}{2} \rfloor$ counter-clockwise from a_j. Since the number of counter-clockwise consecutive agents in $\{b^1_{a_j}, a_j, \ldots, b^2_{a_j}\}$ is greater than $\lceil \frac{n'}{2} \rceil \geq n$, at least one of $b^1_{a_j}, b^2_{a_j}$ is not a duplicated agent. Thus, at round n', the input of each agent in $\{a_1, \ldots, a_d\}$ is already known by at least one agent $\notin \{a_1, \ldots, a_d\}$.

At round $n' - 1$ the cheater does not know the input value of at least one other agent, so it has no incentive to deviate. At round n' for each duplicated agent the cheating agent pretends to be, its input is already known by a non-duplicated agent, which disables the cheater from lying about its input from round n' and on.

Thus, the cheating agent has no incentive to deviate, contradicting our assumption. □

4.2 Coloring

Here, agents are given exact a-priori knowledge of n, i.e., they know the exact value of n at the beginning of the protocol. We present two protocols for Coloring with rational agents, and discuss their properties. In both algorithms, whenever an agent recognizes that another agent has deviated from the protocol, it immediately outputs $\perp$ resulting in the failure of the algorithm.

4.2.1 Tie Breaking

In most algorithms with rational agents, a prominent strategy [4, 17, 12] is to create a neutral mechanism that when two agents' preferences conflict, the mechanism decides which agent gets its preference, and which does not. We refer to such a mechanism as *tie breaking*.

Since agent *id*s are private and agents may cheat about their *id*, they cannot be used to break ties. However, an *orientation* over an edge shared by both agents, achieved without any agent deviating from the protocol that leads to it, can be such a tie breaking mechanism for coloring: whenever neighbors prefer the same color, we break ties according to the orientation of the link between them. Breaking ties for coloring also requires the orientation to be acyclic, since a cycle in which all agents prefer the same color creates a "tie" that isn't broken by the orientation.

Note that since the agents are rational, unless agent a knows that one or more of its neighbors output its preferred color p_a, it will output it itself regardless of the result of the algorithm, which is a *deviation*. Thus, any coloring algorithm must ensure that whenever an agent can output its preferred color, it does, otherwise the agent has an incentive to deviate.

We create an acyclic orientation by a *Renaming* algorithm that reaches equilibrium [4]. The algorithm gives new names $1, \ldots, n$ to the agents, which is in fact an n-coloring of G; however due to the circumstances described above (each agent should output its preference if none of its neighbors does), this n-coloring is not enough. Instead, each agent, in order of the new names, picks its preferred color if available, or the minimal available color otherwise, and sends its color to all of its neighbors.

Algorithm 3 Distributed Coloring Using Renaming (for agent a)

set $T := \varnothing$
Run *Renaming* subroutine
Denote $N(v) \in \mathbb{N}$ the name of $v \in V$ from the Renaming subroutine
Denote t the first round after *Renaming* was complete
for round $i = t$; $i <= t + n; i++$ **do**
    if $N(a) = t + i$ **then**
        if $p_a \notin T$ **then**
            Send p_a to all neighbors
            Set $o_a = p_a$
        else
            Send $m = \min_k k \notin T$
            Set $o_a = m$
    else
        if received $N(v)$ from neighbor v **then**
            $T = T \cup \{N(v)\}$

Theorem 4.2. *Algorithm 3 reaches Distributed Equilibrium for the coloring problem.*

Proof. Let a be an arbitrary agent. Assume in contradiction that at some round r there is a possible step $s \neq s_r$ such that:

$$\mathbb{E}_{s,r}[u_a] > \mathbb{E}_{s_r,r}[u_a]$$

First, it must be shown that an agent does not have an incentive to deviate in the subroutine in order to affect the output of the entire algorithm. In the case of Algorithm 3, the only deviation that would benefit a in the Renaming subroutine is to minimize $N(a)$, i.e. ensuring it picks a color as early as possible. From the building block in [4] we get that the Renaming building block is an equilibrium for agents with preferences on the resulting names, thus ensuring that there is no relevant deviation possible in the subroutine as no agent can unilaterally improve the probability of having a lower N value.

Another property of the Renaming subroutine is that, after its completion, all agents know the names assigned to all agents in the network.

Consider the possible steps a could take at any round r following the Renaming subroutine:

- Sending a message out of order is immediately recognized, as all N values are known to all the agents as well as the round number. i.e., in any round $r \neq t + N(a)$, a has no incentive send any message at all, since it fails the algorithm.
- At $r = t + N(a)$, a must output a color and send it to its neighbors. If $p_a \notin T$ then a outputs p_a. It also has no incentive *not* to correctly notify its neighbors that it is its output, as this notification ensures none of them output p_a (as that would result in 0 utility for that neighbor). If $p_a \in T$ then the color is taken by a neighbor, and a has no incentive to deviate since its utility is already 0.

Thus, the algorithm solves Coloring and is an equilibrium. □

4.2.2 Improving The Algorithm

The *Renaming* process induces more than an acyclic orientation of graph G, it is a total ordering of all agents in the graph. Coloring, however, is in many cases a local property and can be decided

locally [22, 10, 14]. Additionally, the *Renaming* protocol in [4] uses a costly $O(|E| \cdot n^2)$ message complexity.

We present another algorithm for coloring, detailed in Algorithm 6, which improves the message complexity to $O(|E|n)$ by computing an acyclic orientation of graph G.

First, run Wake-Up [4] to learn the graph topology and the *id*s of all agents. Then, in order of *id*s, each agent a draws a random number $S(a)$ with a neighboring "witness" agent $w(a)$ as specified in Algorithm 4, and sends it to all of its neighbors. The number is drawn in the range $1, \ldots, n$ and is different than the numbers of all neighbors of a, which is in fact a coloring of G. However, due to the circumstances described in 4.2.1, this coloring is not enough. By picking a random number with a witness, the agent cannot cheat in the random number generation process, and $w(a)$ is marked as a witness for future verification. When done, each agent simultaneously verifies the numbers published by its neighbors using Algorithm 5, which enables it to receive each value through two disjoint paths: directly from the neighbor, and via the shortest simple path to the neighbor's witness that does not include the neighbor. Then each agent, in order of the values drawn randomly, picks its preferred color if available, or the minimal available color otherwise, and sends its color to all of its neighbors.

The resulting message complexity of the algorithm is as follows: $Wake-Up$ is $O(|E| \cdot n)$. Drawing a random number is called n times and thus uses $O(|E|)$ messages in total, to publish S values to neighbors. Verifying the value of a neighbor uses $O(diameter)$ messages and is called $|E|$ times, for a total of $O(|E| \cdot diameter)$ messages. Sending the output color to all neighbors uses an additional $O(|E|)$ messages. The total number of messages is thus $O(|E| \cdot n)$.

Algorithm 4 Draw(T) Subroutine (for agent a)

Denote $X = 1, ..., n \setminus T$ ▷ X is the set of numbers not drawn by neighbors
1: $w(a) :=$ minimal id $u \in N(a)$
send *witness* to $w(a)$ ▷ choose neighbor with minimal id as witness
2: $r(a) := random\{1, ..., |X|\}$ drawn by a
$r(w(a)) := random\{1, ..., |X|\}$ drawn by $w(a)$
send $r(a)$ to $w(a)$
receive $r(w(a))$ from $w(a)$ ▷ a and witness jointly draw a random number
3: Let $q := r(a) + r(w(a))$ mod $|X|$.
Save $S(a) :=$ q'th largest number in X
send $S(a)$ to all $u \in N(a)$ ▷ Calculate $S(a)$ and publish to neighbors

Algorithm 5 Prompt(u) Subroutine (for agent a)

upon receiving prompt(u) from $u \in N(a)$:
1: $p :=$ shortest simple path $a \to w(a) \to u$
send $S(a), u$ via p ▷ If $v \neq w(a)$ is asked to relay $S(a)$, v fails the algorithm
send $S(a)$ to u via $e = (a, u)$ ▷ u validates that both messages received are consistent

Algorithm 6 Coloring via Acyclic Orientation (for agent a)

1: Run Wake-Up ▷ After which all agents know graph topology
2: set $T := \varnothing$
3: **for** $i = 1, ..., n$ **do**
4: **if** $id_a = i$'th largest id in V **then**
5: $Draw(T)$
6: **else**
7: **wait** $|Draw|$ rounds ▷ Draw takes a constant number of rounds
8: **if** received $S(v)$ from $v \in N(a)$ **then**
9: $T = T \cup \{S(v)\}$ ▷ Add $S(v)$ to set of taken values
10: **for** $u \in N(a)$ **simultaneously do**
11: $Prompt(u)$ ▷ Since we must validate the value received in line 8
12: **wait** until all prompts are completed in the entire graph ▷ At most n rounds
13: **for** round $t = 1, ..., n$ **do**:
14: **if** $S(a) = t$ **then** ▷ Wait for your turn, decreed by your S value
15: **if** $\forall v \in N(a) : o_v \neq p_a$ **then** $o_a := p_a$
16: **else** $o_a :=$ minimum color unused by any $v \in N(a)$
17: **send** o_a to $N(a)$

Theorem 4.3. *Algorithm 6 reaches Distributed Equilibrium for the coloring problem.*

Proof. Let a be an arbitrary agent. Assume in contradiction that at some round r there is a possible step $s \neq s_r$ such that:

$$\mathbb{E}_{s,r}[u_a] > \mathbb{E}_{s_r,r}[u_a]$$

Consider the possible deviations for a in every phase of Algorithm 6:

- Cheating in Wake-Up. The expected utility is independent of the order by which agents draw their random number in Algorithm 4, i.e., the order by which agents initiate Algorithm 4 has no effect on the order by which they will later set their colors. so a has no incentive to publish a false id in the Wake-Up building block.
- Drawing a random number with a witness is an equilibrium: Both agents send a random number at the same round.
- Publishing a false S value will be caught by a future verification process with $w(a)$ when all S values are verified (step 10 of Algorithm 6).
- Sending a color message not in order will be immediately recognized by the neighbors, since S values were verified.
- a might output a different color than the color dictated by Algorithm 6. But if the preferred color is available, then outputting it is the only rational behavior. Otherwise, the utility for the agent is already 0 in any case.

Thus, the algorithm solves Coloring and is an equilibrium. □

5 How Much Knowledge Is Necessary?

Here we examine the effects of a-priori knowledge that *bound* the possible value of n. We show that the possibility of algorithms that are equilibria depends on the range $[\alpha, \beta]$ in which n might be, and show these ranges for different problems.

We show that different problems that we examine in this paper require different bounds. For some problems, there is an equilibrium given *any* bound; for other problems, there is no equilibrium unless tight bounds are given. This leads us to classify problems by the relations between the upper and lower bounds a-priori given on n necessary and sufficient to find equilibrium for these problems.

Table 1 summarizes our results. Partition and Orientation have equilibria without any knowledge of n; however, the former is constrained to even-sized rings, and the latter is a trivial problem in distributed computing (radius 1 in the *LOCAL* model [21]).

Definition 5.1 ((α, β)-Knowledge)**.** We say agents have (α, β)-*Knowledge* about the actual number of agents n, $\alpha \leq \beta$, if all agents know that the value of n is in $[\alpha, \beta]$. We assume agents have no information about the distribution over $[\alpha, \beta]$, i.e., they assume it is uniform.

Definition 5.2 (f-Bound)**.** Let $f : \mathbb{N} \to \mathbb{N}$. A distributed computing problem $\mathbb{P}$ is f-*bound* if:

- There exists an algorithm for $\mathbb{P}$ that is an equilibrium given (α, β)-Knowledge for any α, β such that $\beta \leq f(\alpha)$.
- For any algorithm for $\mathbb{P}$, there exist α, β where $\beta > f(\alpha)$ such that given (α, β)-Knowledge the algorithm is not an equilibrium.

In other words, a problem is f-bound if given (α, β)-Knowledge, there is an equilibrium when $\beta \leq f(\alpha)$, and there is no equilibrium when $\beta > f(\alpha)$. A problem is ∞-bound if there is an equilibrium given *any* bound f, but there is no equilibrium with $(1, \infty)$-Knowledge. A problem is *unbounded* if there is an equilibrium with $(1, \infty)$-Knowledge.

Bound	Problem (in a ring)
$\alpha + 1$	Leader Election[7]
$2\alpha - 2$	Knowledge Sharing
∞	Coloring, 2-Knowledge Sharing
unbounded	Partition, Orientation[7]

Table 1: Knowledge Bounds in a Ring; summary of results

Consider an agent a at the start of a protocol given (α, β)-Knowledge. If a pretends to be a group of d agents, it can be caught when $d + n - 1 > \beta$, since agents might discover the number of agents and catch the cheater. Moreover, *any* duplication now involves some risk since the actual value of n is not known to the cheater.

An arbitrary cheating agent a simulates executions of the algorithm for every possible duplication, and evaluates its expected utility. Denote D a duplication scheme in which an agent pretends to be d agents. Let $P_D = P[d + n - 1 \leq \beta]$ be the probability, from agent a's perspective, that the overall size does not exceed β. If for agent a there exists a duplication scheme D at round 0 such that $\mathbb{E}_{D,0}[u_a] \cdot P_D > \mathbb{E}_{s(0),0}[u_a]$, then agent a has an incentive to deviate and duplicate itself.

For each problem we look for the maximal range of α, β where no d exists that satisfies the inequation above.

[7] These results hold in general graphs, as well.

Hereafter, we classify Knowledge Sharing (5.1) and Coloring in a ring (5.2) into the hierarchy. Additionally, we classify Leader Election (5.3), Ring Partition (5.4) and Orientation (5.5) as extreme cases. The former is $(\alpha+1)$-bounded, while the latter two are unbounded, but either have some other form of knowledge about n, or are very local.

5.1 Knowledge Sharing

Theorem 5.3. *Knowledge Sharing is* $(2\alpha-2)$*-bound.*

Corollary 5.4. 2*-Knowledge Sharing is* ∞*-bound.*

Proof. Assume agents have (α,β)-knowledge for some α,β. A cheating agent a's goal is to choose a value d, the number of agents it pretends to be, that maximizes its expected utility.

Let k be the number of possible outputs of a Knowledge Sharing algorithm, i.e., the range of the output function g is of size k. By the Full Knowledge property (definition 2.4), any output is *equally* possible. Therefore, without deviation the expected utility of a at round 0 is: $\mathbb{E}_{s(0),0}[u_a]=\frac{1}{k}$.

According to Theorem 4.1, Algorithm 2 is an equilibrium for Knowledge Sharing in a ring when a cheating agent pretends to be n agents or less. Corollary 3.2 shows that when a cheating agent pretends to be *more* than n agents, no algorithm for Knowledge Sharing is an equilibrium. Thus, looking at all possible values of n in the range $[\alpha,\beta]$, a wants to maximize the probability that $d>n$ and the duplication increases its utility, while also minimizing the probability that $d+n-1>\beta$ and the algorithm fails.

If a cheats and pretends to be d agents, then necessarily $d\geq\alpha$, otherwise according to Theorem 4.1 there is an equilibrium, and the duplication does not increase its utility for any value of n in $[\alpha,\beta]$. Additionally it holds that $d\leq\lceil\frac{\beta}{2}\rceil+1$, since any higher value of d increases a's chances of being caught and failing the algorithm (when $d+n-1>\beta$), without increasing the number of possible values of n for which its utility is higher (when $d>n$ and $d+n-1\leq\beta$).

From a's perspective at the beginning of the algorithm, the value of n is uniformly distributed over $[\alpha,\beta]$, i.e., there are a total of $\beta-\alpha+1$ equally possible values for n. According to Corollary 3.2 when $d>n$ agent a can increase its expected utility by deviating. Let $X>\frac{1}{k}$ be the utility a gains by pretending to be $d>n$ agents successfully (i.e., when $d+n-1\leq\beta$ and the algorithm does not fail). Since n is uniformly distributed over $[\alpha,\beta]$ this has a probability of $\frac{d-\alpha}{\beta-\alpha+1}$ to occur. On the other hand, when pretending to be $d\leq n$ agents successfully the utility of a does not change and is $\frac{1}{k}$, and this has a probability of $\frac{\lceil\frac{\beta}{2}\rceil-d+1}{\beta-\alpha+1}$. In all other cases $d+n-1>\beta$ and the algorithm fails, resulting in a utility of 0. Thus, the expected utility of agent a at round 0:

$$\mathbb{E}_{D,0}[u_a]=X\cdot\frac{d-\alpha}{\beta-\alpha+1}+\frac{1}{k}\cdot\frac{\lceil\frac{\beta}{2}\rceil-d+1}{\beta-\alpha+1} \tag{2}$$

By the constraints specified above, the value of d that maximizes (2) is $d=\lfloor\frac{\beta}{2}\rfloor+1$, and a will deviate from the algorithm whenever $\mathbb{E}_{D,0}[u_a]>\frac{1}{k}$. To find the f-bound on $[\alpha,\beta]$ we derive β as a function of α:

$$\mathbb{E}_{D,0}[u_a]=X\cdot\frac{\lfloor\frac{\beta}{2}\rfloor+1-\alpha}{\beta-\alpha+1}+\frac{1}{k}\cdot\frac{\lceil\frac{\beta}{2}\rceil-\lfloor\frac{\beta}{2}\rfloor}{\beta-\alpha+1}>\frac{1}{k}$$

$$\begin{cases}\beta\text{ is even} & \beta(kX-2)>2\alpha kX-2kX-2\alpha+2\\ \beta\text{ is odd} & \beta(kX-2)>2\alpha kX-kX-2\alpha\end{cases} \tag{3}$$

From (3) we can derive the following:

1. The inequations are satisfiable only if $X > 2 \cdot \frac{1}{k}$. Since $X \leq 1$, 2-Knowledge Sharing ($k = 2$) cannot satisfy the inequations and a never has an incentive to deviate, given *any* bound. This proves Corollary 5.4.

2. For Knowledge Sharing, we find the range $[\alpha, \beta]$ that holds for any k. As k grows large, $\frac{1}{k}$ nears 0. Assuming the profit when duplication is successful is $X = 1$, agent a has an incentive to deviate when $\lfloor \frac{\beta}{2} \rfloor + 1 - \alpha > 0$. When β is even: $\beta > \alpha - 2$, and when β is odd: $\beta > \alpha - 1$. Thus, Algorithm 2 is an equilibrium for Knowledge Sharing when agents have (α, β)-knowledge such that $\beta \leq \alpha - 2$, and for any algorithm for Knowledge Sharing there exist $\alpha, \beta > \alpha - 2$ such that there is no equilibrium when agents have (α, β)-knowledge. This proves Theorem 5.3.

□

5.2 Coloring

Theorem 5.5. *Coloring in a ring is ∞-bound.*

Proof. Consider Algorithm 7 which solves coloring in a ring using 2-Knowledge Sharing.

Algorithm 7 Coloring in a Ring

1: Wake-Up to learn the size of the ring
2: Assume an arbitrary global direction over the ring (this can be relaxed via Leader Election [4])
3: Run 2-Knowledge Sharing to randomize a single global bit $b \in \{0, 1\}$
4: Publish the preferred color of each agent simultaneously over the entire ring
5: In each group of consecutive agents that prefer the same color, if $b = 0$ the even agents (according to the orientation) output their preferred color, else the odd agents do.
6: If an agent has no neighbors who prefer the same color, it outputs its preferred color.
7: Any other agent outputs the minimal available color.

It is easy to see that Algorithm 7 is an equilibrium and results in a legal coloring of the ring. It uses 2-Knowledge Sharing and thus, following Corollary 5.4, it proves Theorem 5.5. □

5.3 Leader Election

In the Leader Election problem, each agent a outputs $o_a \in \{0, 1\}$, where $o_a = 1$ means that a was elected leader and $o_a = 0$ means otherwise. $\Theta_L = \{O | \exists a : o_a = 1, \forall b \neq a : o_b = 0\}$. We assume that every agent prefers either 0 or 1.

Theorem 5.6. *Leader Election is $(\alpha + 1)$-bound.*

Proof. Recall that any Leader Election algorithm must be *fair* [3], i.e., every agent must have equal probability of being elected leader for the algorithm to be an equilibrium.

Given $f(\alpha) = \alpha + 1$, the actual number of agents n is either α or $\alpha + 1$, decided by some distribution unknown to the agents. If an agent follows the protocol, the probability of being elected is $\frac{1}{n}$. If it duplicates itself once, the probability that a duplicate is elected is $\frac{2}{n+1}$, but if $n = \alpha + 1$ the protocol fails and the utility is 0. Thus $\mathbb{E}_{s_{dup}, a}[u_a] = \frac{1}{2}\frac{2}{n+1} < \frac{1}{n}$, i.e., no agent has an incentive to deviate.

Given $f(\alpha) = \alpha + 2$, then n is in $[\alpha, \alpha + 2]$. If an agent follows the protocol, its expected utility is still $\frac{1}{n}$. If it duplicates itself once, the probability that a duplicate is elected is still $\frac{2}{n+1}$, however

only if $n = \alpha + 2$ the protocol fails. Thus, $\mathbb{E}_{s_{dup},a}[u_a] = \frac{2}{3}\frac{2}{n+1} > \frac{1}{n}$ for any $n > 3$. So the agent *has* an incentive to deviate.

Thus for $f(\alpha) = \alpha + 1$ the algorithm presented in [3] is an equilibrium, while for $f(\alpha) = \alpha + 2$ no algorithm for Leader Election is an equilibrium, since any algorithm must be *fair*. □

5.4 Ring Partition

In the Ring Partition problem, the goal is to partition the agents of an even-sized ring into two, equally-sized groups: group 0 and group 1. We assume that every agent prefers to belong to either group 0 or 1.

By definition, the Ring Partition problem is defined only for graphs of even size. Formally, let $sum_x = |\{i|o_i = x\}|$, then we define Θ_L in the following manner:

$$\Theta_L = \begin{cases} sum_0 = sum_1 \text{ and } sum_\perp = 0 & n \text{ is even} \\ \varnothing & n \text{ is odd} \end{cases}$$

Theorem 5.7. *Ring Partition is unbounded.*

Proof. It is clear that an agent will not duplicate itself to change the parity of the graph, as that will necessarily cause an erroneous output. So it is enough to show an algorithm that is an equilibrium for even graphs, when agents have no knowledge about n. Consider the following algorithm:

- Either one arbitrary agent wakes up or we run a Wake-Up subroutine and then Leader Election [4]. Since the initiator (leader) has no effect on the output, both are an equilibrium in this application.
- The initiating agent sends a token which alternatively marks agents by 0 or 1 and also defines the direction of communication in the ring.
- Upon reception of the token with value m, an agent a does one of the following:
  1. If $m = 0$, send predecessor (denoted $p(a)$) a random bit t_a.
  2. Else, if $m = 1$, wait for 1 round and send successor (denoted $s(a)$) a random bit t_a.
- Upon reception of the neighbor's bit (one round after receiving the token), set
$$o_a = (t_a + t_{s(a)/p(a)} + m)_{mod2}$$
- As the token arrives back at the initiator, it checks the token's parity. For even rings, it must be the opposite value from the value it originally sent.

This algorithm divides every pair of agents to one with output 1 and one with output 0, as the token value m is different, thus achieving a partition.

We show that it is also an equilibrium. Assume an agent a deviates at some round r. If r is in the Wake-Up or Leader Election phase in order to be the initiator, it cannot improve its utility since choosing the starting value of the token, choosing the direction, or being first cannot increase the agent's utility. If it is a deviation while the token traverses other parts of the graph, any message a sends will eventually be discovered, as the real token has either already passed or will eventually pass through the "cheated" agent. If a changes the value of the token, a randomization between two agents will be out of sync eventually at the end of the token traversal, and also the initiator will recognize that the ring does not contain an even number of agents. During the exchange of t_a the result is independent of a's choice of value for t_a. So there is no round in which a can deviate from the protocol. □

5.5 Orientation

In the Orientation problem the two ends of each edge must agree on a direction for this edge. We assume that every agent prefers certain directions for its edges.

Unlike Ring Partition, Orientation is defined for any graph. It is, however, a very local problem (radius 1 in the *LOCAL* model [21]).

Theorem 5.8. *The Orientation problem is unbounded.*

Proof. We show a simple algorithm and prove that it is an equilibrium without any a-priori knowledge of n or bounds on n. Assuming all agents start at the same round (otherwise run Wake-Up), consider the following algorithm:

- Each agent simultaneously send a random number $(0, 1)$ and its *id* on each of its edges.
- For each edge, XOR the bit you sent and the bit received over that edge
- If the result is 1, the edge is directed towards the agent with the higher *id*, otherwise it is directed towards the lower *id*.
- Every agent outputs the list of pairs with *id* and direction for each of its neighbors.

Since an agent's utility is defined over its personal output, Solution Preference inhibits agents to output a correct set of pairs, so a cheater may only influence the direction of the edges. Since duplication does not create any new edges between the cheater and the original graph, and the orientation is decided over each edge independently, it does not effect any agent's utility. Other than that, randomizing a single bit over an edge at the same round is in equilibrium. So the algorithm is an equilibrium, and Orientation is unbounded. □

6 Discussion

Distributed algorithms are commonly required to work in an arbitrarily large network. In a realistic scenario, the exact size of the network may not be known to all of its members. In this paper, we have shown that in most problems the use of duplication gives an agent power to affect the outcome of the algorithm. The amount of duplications an agent can create is limited by the ability of other agents to detect this deviation, and the only tool for this ability is a-priori knowledge about n. Section 3 shows that for some problems, without any such knowledge, distributed problems become impossible to solve without any agent having an incentive to deviate from the algorithm.

The f-bounds we proved for common distributed problems show that the initial knowledge required for equilibrium to be possible depends on the balance between two factors: (1) The amount of duplications necessary to increase an agent's expected utility and by how much it increases, and (2) the expected utility for an agent if it follows the protocol. In order for an agent to have an incentive to duplicate itself, an undetected duplication needs to be either a lot more profitable than following the algorithm or it must involve low risk of being caught.

Our results produce several directions that may be of interest:

1. Proving impossibility and f-bounds in general topology graphs, as for some of the problems we only discussed ring networks.
2. Proving impossibility and showing algorithms for other problems with rational agents, which result in other tight f-bounds.

3. Finding a problem that is α-bound, i.e., has an equilibrium only when n is known exactly.

4. What defines a trivial or non-trivial problem with rational agents? More specifically, finding a characteristic that separates problems that can be solved without *any* knowledge about n from ones in which at least some bounds must be a-priori known.

5. Finding an *unbounded* problem not inherently limited (as Orientation or ring Partition are), or finding proof that no such problem exists.

6. Exploring the effects of initial knowledge about network size in an asynchronous setting.

7 Acknowledgment

We would like to thank Doron Mukhtar for showing us the ring partition problem and proving it is *unbounded*, when we thought such problems do not exist. We would also like to thank Michal Feldman, Amos Fiat, and Yishay Mansour for helpful discussions.

References

[1] I. Abraham, D. Dolev, R. Gonen, and J. Y. Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *PODC*, pages 53–62, 2006.

[2] I. Abraham, D. Dolev, and J. Y. Halpern. Lower bounds on implementing robust and resilient mediators. In *TCC*, pages 302–319, 2008.

[3] I. Abraham, D. Dolev, and J. Y. Halpern. Distributed protocols for leader election: A game-theoretic perspective. In *DISC*, pages 61–75, 2013.

[4] Y. Afek, Y. Ginzberg, S. Landau Feibish, and M. Sulamy. Distributed computing building blocks for rational agents. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 406–415, New York, NY, USA, 2014. ACM.

[5] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. Bar fault tolerance for cooperative services. In *SOSP*, pages 45–58, 2005.

[6] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.

[7] B. Awerbuch, M. Luby, A. V. Goldberg, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, pages 364–369, Washington, DC, USA, 1989. IEEE Computer Society.

[8] I. Bárány. Fair distribution protocols or how the players replace fortune. *Math. Oper. Res.*, 17(2):327–340, May 1992.

[9] E. Ben-Porath. Cheap talk in games with incomplete information. *J. Economic Theory*, 108(1):45–71, 2003.

[10] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Inf. Control*, 70(1):32–53, July 1986.

[11] V. Dani, M. Movahedi, Y. Rodriguez, and J. Saia. Scalable rational secret sharing. In *PODC*, pages 187–196, 2011.

[12] Y. Dodis, S. Halevi, and T. Rabin. A cryptographic solution to a game theoretic problem. In *CRYPTO*, pages 112–130, 2000.

[13] G. Fuchsbauer, J. Katz, and D. Naccache. Efficient rational secret sharing in standard communication networks. In *TCC*, pages 419–436, 2010.

[14] A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 315–324, New York, NY, USA, 1987. ACM.

[15] S. D. Gordon and J. Katz. Rational secret sharing, revisited. In *SCN*, pages 229–241, 2006.

[16] A. Groce, J. Katz, A. Thiruvengadam, and V. Zikas. Byzantine agreement with a rational adversary. In *ICALP (2)*, pages 561–572, 2012.

[17] J. Y. Halpern and X. Vilaça. Rational consensus: Extended abstract. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 137–146, New York, NY, USA, 2016. ACM.

[18] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 7–15, New York, NY, USA, 2006. ACM.

[19] M. Lepinski, S. Micali, C. Peikert, and A. Shelat. Completely fair sfe and coalition-safe cheap talk. In *PODC*, pages 1–10, 2004.

[20] N. Linial. Legal coloring of graphs. *Combinatorica*, 6(1):49–54, 1986.

[21] N. Linial. Distributive graph algorithms global solutions from local data. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 331–335, Washington, DC, USA, 1987. IEEE Computer Society.

[22] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[23] A. Lysyanskaya and N. Triandopoulos. Rationality and adversarial behavior in multi-party computation. In *CRYPTO*, pages 180–197, 2006.

[24] R. McGrew, R. Porter, and Y. Shoham. Towards a general theory of non-cooperative computation. In *TARK*, pages 59–71, 2003.

[25] T. Moscibroda, S. Schmid, and R. Wattenhofer. When selfish meets evil: byzantine players in a virus inoculation game. In *PODC*, pages 35–44, 2006.

[26] M. Osborne and A. Rubinstein. *A Course in Game Theory*. A Course in Game Theory. MIT Press, 1994.

[27] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[28] Y. Shoham and M. Tennenholtz. Non-cooperative computation: Boolean functions with correctness and exclusivity. *Theoretical Computer Science*, 343(12):97 – 113, 2005.

[29] M. Szegedy and S. Vishwanathan. Locality based graph coloring. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 201–207, New York, NY, USA, 1993. ACM.

[30] A. Urbano and J. E. Vila. Computational complexity and communication: Coordination in two-player games. *Econometrica*, 70(5):1893–1927, September 2002.

[31] A. Urbano and J. E. Vila. Computationally restricted unmediated talk under incomplete information. *Economic theory*, 2004.

[32] E. L. Wong, I. Levy, L. Alvisi, A. Clement, and M. Dahlin. Regret freedom isn't free. In *OPODIS*, pages 80–95, 2011.