Distributed anonymous function computation in information fusion and multiagent systems

The MIT Faculty has made this article openly available. Please share how this access benefits you. Your story matters.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>As Published</td>
<td><a href="http://dx.doi.org/10.1109/ALLERTON.2009.5394487">http://dx.doi.org/10.1109/ALLERTON.2009.5394487</a></td>
</tr>
<tr>
<td>Publisher</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>Version</td>
<td>Final published version</td>
</tr>
<tr>
<td>Accessed</td>
<td>Sun Jan 13 08:36:05 EST 2019</td>
</tr>
<tr>
<td>Citable Link</td>
<td><a href="http://hdl.handle.net/1721.1/59370">http://hdl.handle.net/1721.1/59370</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>Article is made available in accordance with the publisher’s policy and may be subject to US copyright law. Please refer to the publisher’s site for terms of use.</td>
</tr>
<tr>
<td>Detailed Terms</td>
<td></td>
</tr>
</tbody>
</table>
Distributed anonymous function computation in information fusion and multiagent systems
Julien M. Hendrickx, Alex Olshevsky, John N. Tsitsiklis

Abstract—We propose a model for deterministic distributed function computation by a network of identical and anonymous nodes, with bounded computation and storage capabilities that do not scale with the network size. Our goal is to characterize the class of functions that can be computed within this model. In our main result, we exhibit a class of non-computable functions, and prove that every function outside this class can at least be approximated. The problem of computing averages in a distributed manner plays a central role in our development.

I. INTRODUCTION

The goal of many multi-agent systems, distributed computation algorithms and decentralized data fusion methods is to have a set of nodes compute a common value based on initial values or observations at each node. Towards this purpose, the nodes, which we will sometimes refer to as agents, perform some internal computations and repeatedly communicate with each other. Let us consider some examples.

(a) Quantized consensus: Suppose that each agent begins with an integer value \( x_i(0) \in \{0, \ldots, K\} \). We would like the agents to end up, at some later time, with values \( y_i \) that are almost equal, i.e., \( |y_i - y_j| \leq 1 \), for all \( i, j \), while preserving the sum of the values, i.e., \( \sum_{i=1}^{n} x_i(0) = \sum_{i=1}^{n} y_i \). This is the so-called quantized averaging problem which has received considerable attention recently; see [16], [9], [3], [20]. It may be viewed as the problem of computing the function \((1/n) \sum_{i=1}^{n} x_i\), rounded to the nearest integer.

(b) Distributed hypothesis testing: Consider \( n \) sensors interested in deciding between two hypotheses, \( H_0 \) and \( H_1 \). Each sensor collects measurements and makes a preliminary decision \( x_i \in \{0, 1\} \) in favor of one of the hypotheses. The sensors would like to make a final decision by majority vote, in which case they need to compute the indicator function of the event \( \sum_{i=1}^{n} x_i \geq n/2 \), in a distributed way. Alternatively, in a weighted majority vote, they may be interested in computing the indicator function of the event \( \sum_{i=1}^{n} x_i \geq 3n/4 \).

(c) Solitude verification: This is the problem of verifying that at most one node in the network has a given state.

This problem is of interest if we want to avoid simultaneous transmissions over a common channel [13], or if we want to maintain a single leader (as in motion coordination — see for example [15]) Given \( K \) possible states, so that \( x_i \in \{1, \ldots, K\} \), solitude verification is equivalent to the problem of computing the binary function which is equal to 1 if and only if \( \{|i: x_i = 1|\} = 1 \).

There are numerous methods that have been proposed for solving problems such as the above. (See for example the vast and growing literature on consensus and averaging methods.) Oftentimes, different algorithms involve different computational capabilities on the part of the agents, which makes it hard to talk about “the best” algorithm. At the same time, simple algorithms (such as setting up a spanning tree and aggregate information by progressive summations over the tree) are often “disqualified” because they require too much coordination or global information. One then realizes that a sound discussion of such issues requires the specification of a precise model of computation, followed by a systematic analysis of fundamental limitations under any given model.

This is precisely the objective of this paper: to propose a particular model, and to characterize the class of computable functions under this model.

Our model provides an abstraction for the most common requirements for distributed algorithms in the sensor network literature. It is somewhat special because (i) it does not allow for randomization; (ii) it does not address the case of time-varying interconnection graphs; such extensions are left for future research. Qualitatively speaking, our model includes the following features.

Identical agents: Any two agents with the same number of neighbors must run the same algorithm.

Anonymity: An agent can distinguish its neighbors using its own, private, local identifiers. However, agents do not have global identifiers.

Absence of global information: Agents have no global information, and do not even have an upper bound on the total number of nodes. Accordingly, the algorithm that each agent is running is independent of the network size and topology.

Convergence: Agents hold an estimated output, and this estimate must converge to a desired value which is generally a function of all agents’ initial observations or values. In particular, for the case of discrete outputs, all agents must eventually settle on the desired value. On the other hand, the agents do not need to be aware of such termination, which is anyway impossible in the absence of any global information [6].
A. Goal and Contribution

We provide in this paper a general model of decentralized anonymous computation with the above described features, and characterize the type of functions of the initial values that can be computed. To keep our model simple, we only consider deterministic and synchronized agents exchanging messages on a fixed bi-directional network, with no time-delays or unreliable transmissions. Agents are modelled as finite automata, so that their individual capabilities remain bounded as the number of agents increases.

We prove that if a function is computable under our model, then its value only depends on the frequencies of the different possible initial values. For example, if the initial values \( x_i \) only take values 0 and 1, a computable function necessarily only depends on \( p_0 := |\{i : x_i = 0\}|/n \) and \( p_1 := |\{i : x_i = 1\}|/n \). In particular, determining the number of nodes, or whether at least two nodes have an initial value of 1 is impossible.

Conversely, we prove that if a function only depends on the frequencies of the different possible initial values (and is measurable), then the function can at least be approximated with any given precision, except possibly on a set of frequency vectors of arbitrarily small volume. Moreover, if the dependence on these frequencies can be expressed by a combination of linear inequalities with rational coefficients, then the function is computable exactly. In particular, the functions involved in the quantized consensus and distributed hypothesis testing examples are computable, whereas the function involved in solitude verification is not. Similarly, statistical measures such as the standard deviation and the kurtosis can be approximated with arbitrary precision.

Finally, we show that with infinite memory, the frequencies of the different values (i.e., \( p_0, p_1 \) in the binary case) are computable.

B. Overview of previous work

There is a large literature on distributed function computation in related models of computation. A common model in the distributed computing literature involves the requirement that all processes terminate when the desired output is produced. A consequence of the termination requirement is that nodes typically need to know the network size \( n \) (or an upper bound on \( n \)) to compute any non-constant functions. We refer the reader to [1], [6], [31], [17], [26] for some fundamental results in this setting, and to [10] for an excellent summary of known results.

Similarly, the biologically-inspired “population algorithm” model of distributed computation has some features in common with our model, namely finite-size agents and lack of a termination condition; see [2] for a very nice summary of known results. However, these models involve a somewhat different type of agent interactions from the ones we consider.

We note that the impossibility of computing \( p_1 \) without any memory was shown in [21]. Some experimental memoryless algorithms were proposed in the physics literature [12]. Randomized algorithms for computing particular functions were investigated in [16], [7]. We also point the reader to the literature on “quantized averaging,” which often tends to involve similar themes [9], [20], [3], [8].

Several papers quantified the performance of simple heuristics for computing specific functions, typically in randomized settings. We refer the reader to [14] and [29], which studied simple heuristics for computing the majority function. A deterministic algorithm for computing the majority function (and some more generalized functions) was proposed in [23].

Semi-centralized versions of the problem, in which the nodes ultimately transmit to a fusion center, have often been considered in the literature, e.g., for distributed statistical inference [25] or detection [19]. The papers [11], [18], and [22] consider the complexity of computing a function and communicating its value to a sink node. We refer the reader to the references therein for an overview of existing results in such semi-centralized settings.

Our results differ from previous works in several key respects: (i) Our model, which involves totally decentralized computation, deterministic algorithms, and constraints on memory and computation resources at the nodes, does not require the nodes to know when the computation is over, does not seem to have been studied before. (ii) Our focus is on identifying computable and non-computable functions under our model, and we achieve a nearly tight separation.

II. Formal description of the model

The system consists of (i) a communication graph \( G = (V, E) \), which is bidirectional (i.e., if \((i,j) \in E\), then \((j,i) \in E)\); (ii) a port labeling whereby edges outgoing from node \( i \) are labeled by \( port\) \( n_{o_d, i} \) in the set \(\{1, 2, \ldots, deg(i)\}\); (iii) a family of finite automata \( (A_{0, d})_{d=1,2,3,\ldots} \) (The automaton \( A_d \) is meant to describe the behavior of a node with degree \( d \)).

The state of the automaton \( A_d \) is a tuple \((x, z, y, m_1, \ldots, m_d) \); we will call \( x \in X = \{0, 1, \ldots, K\} \) the initial value, \( z \in Z_d \) the internal memory state, \( y \in Y \) the output or estimated answer, and \( m_1, \ldots, m_d \in M \) the messages. The sets \( X, Y, Z_d, M \) are assumed finite, unless there is a statement to the contrary. Furthermore, we assume that the number of bits that can be stored at a node is proportional to the node’s degree; that is, \( \log |Z_d| \leq C_d \), for some absolute constant \( C \). (Clearly, at least this much memory would be needed to be able to store the messages received at the previous time step.) We will also assume that \( \emptyset \) is an element of the above defined sets \( Y, Z_d \) and \( M \). The transition law \( A_d \) maps \( X \times Z_d \times Y \times M^d \) to \( X \times Z_d \times Y \times M^d ; (x, z, y; (m_1, \ldots, m_d)) \) is mapped to \([x, z', y'; (m'_1, \ldots, m'_d)] \) . In words, the automaton creates a new memory state, output, and (outgoing) messages at each iteration, but does not change the initial value.

We construct a dynamical system out of the above elements as follows. Let \( d(i) \) be the degree of node \( i \). Node \( i \) begins with an initial value \( x_i \in X \); it implements the automaton \( A_{d(i)} \) initialized with \( x = x_i \) and with \( z = y = m_1 = \ldots = m_d = \emptyset \). We use \( S_i(t) = \).
[xi, yi(t), zi(t), m1i(t), ..., mi, d(i)(t)] to denote the state of the automaton implemented by agent i at round t. Let f1, ..., f d(i) be an enumeration of the neighborhoods of i, and let pk be the port number of the link (jk, i). The evolution of the system is then described by

\[ [xi, zi(t + 1), yi(t + 1); m1i(t + 1), ..., mi, d(i)(t + 1)] = A_{d(i)} [xi, zi(t), yi(t); m_{j1}, p1(t), ..., m_{jd}, p_{d(i)}(t)] . \]

In words, the messages “sent” by the neighbors of i into ports leading to i are used to transition to a new state and create new messages that i “sends” to its neighbors at the next round. We say that y* ∈ Y is the final output of this dynamical system if there is a time t’ such that y_i(t) = y* for every i and t ≥ t’.

Consider now a family of functions \((f_n)_{n \in N} : X^n \rightarrow Y\) for every x ∈ X^n. For every \((x_1, ..., x_n)\), \(f(x_1, ..., x_n) = h(p_1(x_1, ..., x_n), ..., p_K(x_1, ..., x_n))\).

We run our distributed algorithm on the network \(G = \{E, V\}\) with n nodes, any port labelling, and any set of initial conditions \(x_1, ..., x_n\), the final output of the system is always \(f_n(x_1, ..., x_n)\).

In some results, we will also refer to function families \((f_n)_{n \in N}\) computable with infinite memory, by which we mean that the internal memory sets \(Z_d\) and output set Y are countable, the rest of the model being unaffected.

We study in the sequel the general function computation problem: What families of functions are computable, and how can we design the automata \(A_d\) to compute them?

### III. Necessary Condition for Computability

Let us first state the following lemma which can easily be proved by induction on time.

**Lemma III.1.** Suppose that \(G = \{\{1, ..., n\}, E\}\) and \(G' = \{\{1, ..., n\}, E'\}\) are isomorphic, that is, there exists a permutation \(\pi\) such that \((i, j) \in E\) if and only if \((\pi(i), \pi(j)) \in E'\). Further, suppose that the port label at node i for the edge leading to j in G is the same as the port label at node \(\pi(i)\) for the edge leading to \(\pi(j)\) in \(G'\). Then, the state \(S_i(t)\) resulting from the initial values \(x_1, ..., x_n\) with the graph G is the same as the state \(S_{\pi(i)}(t)\) resulting from the initial values \(x_{\pi(1)}, ..., x_{\pi(n)}\) with the graph \(G'\).

**Proposition III.1.** Suppose that the family \(\{f_1(x_1), f_2(x_1, x_2), f_3(x_1, x_2, x_3), \ldots\}\) is computable with infinite memory. Then, each \(f_i\) is invariant under permutations of its arguments.

**Proof:** Let \(\pi_{ij}\) be permutation that swaps i with j; with a slight abuse of notation, we also denote by \(\pi_{ij}\) the mapping from \(X^n\) to \(X^n\) that swaps the i-th and j-th elements of a vector. We show that for all \(x \in X^n\), \(f_n(x) = f_n(\pi_{ij}(x))\).

We run our distributed algorithm on the n-node complete graph. Consider two different initial configurations: (i) starting with the vector x; (ii) starting with the vector \(\pi_{ij}(x)\). Let the way each node enumerates his neighbors in case (i) be arbitrary; in case (ii), let the enumeration be such that the conditions in Lemma III.1 are satisfied, which is easily accomplished. Since the limiting value of \(y_i\) in one case is \(f(x)\) and in the other is \(f(\pi_{ij}(x))\), we obtain \(f(x) = f(\pi_{ij}(x))\). Since the permutations \(\pi_{ij}\) generate the group of permutations, permutation invariance follows.

Let \(x \in X^n\). We will denote by \(x^k\) the concatenation of \(x\) with itself, and, generally, \(x^k\) the concatenation of \(k\) copies of \(x\). We now prove that self-concatenation does not affect the value of a computable family of functions.

**Proposition III.2.** Suppose that the family \(\{f_1(x_1), f_2(x_1, x_2), f_3(x_1, x_2, x_3), \ldots\}\) is computable with infinite memory. Then, for every \(m \geq 2\), every sequence \(x \in X^m\), and every positive integer \(k\), \(f_m(x) = f_{km}(x^k)\).

**Proof:** Consider a ring of agents of size \(m\), where the \(i\)-th agent counterclockwise begins with the \(i\)-th element of \(x\); and consider a ring of size \(km\) where the agents \(i, i + m, i + 2m, \ldots \) (counterclockwise) begin with the \(i\)-th element of \(x\). Suppose further that each node enumerates its two neighbors so that the neighbor on the left is labelled 1, while the neighbor on the right is labelled 2. See Figure 1 for an example with \(m = 3, k = 2\) and \(x_1 = i\).

![Fig. 1. Example of two situations that are indistinguishable by the nodes.](image-url)
where
\[ p_k(x_1, \ldots, x_n) = \frac{|\{i \mid x_i = k\}|}{n}, \]
so that \( p_k(x_1, \ldots, x_n) \) is the frequency of occurrence of the initial value \( k \). In this case, we say that the family \( (f_n) \) is proportion-based.

**Theorem III.1.** Suppose that the family \( (f_n) \) is computable with infinite memory. Then, this family is proportion-based.

**Proof:** Let \( x \) and \( y \) be two sequences of \( n \) and \( m \) elements, respectively, such that \( p_k(x_1, \ldots, x_n) = p_k(y_1, \ldots, y_m) = \hat{p}_k \) for \( k = 1, \ldots, K \), that is, the number of occurrences of \( k \) in \( x \) and \( y \) is \( np_k \) and \( m \hat{p}_k \), respectively. Observe that for any \( k \in X \), the vectors \( x^n \) and \( y^n \) have the same number \( mn \) of elements, and both contain \( mn \hat{p}_k \) occurrences of \( k \). The sequences \( y^n \) and \( x^m \) can thus be obtained from each other by a permutation, which by Proposition III.1 implies that \( f_{mn}(x^m) = f_{mn}(y^n) \). It then follows from Proposition III.2 that \( f_{mn}(x^m) = f_n(x) \) and \( f_{mn}(y^n) = f_n(y) \), so that \( f_n(x) = f_n(y) \). This proves that the value of \( f_n(x) \) is determined by the vector \((p_1(x), \ldots, p_n(x))\).

The following examples illustrate this result.

(a) The parity function \( \sum_{i=1}^n x_i \equiv 0 \mod k \) is not computable, for any \( k > 0 \).

(b) In a binary setting \( (X = \{0, 1\}) \), checking whether the number of nodes with \( x_i = 1 \) is at least \( 10 \) plus the number of nodes with \( x_i = 0 \) is not computable.

(c) Solitude verification, i.e. checking whether \( |\{i \mid x_i = 0\}| = 1 \), is not computable.

(d) An aggregate difference functions such as \( \sum_{i<j} |x_i - x_j| \) is not computable, even calculated modulo \( k \).

**IV. Reduction of Generic Functions to the Computation of Averages**

In this section, we show that the computability question for large classes of functions reduces to the computability question for a particular averaging-like function. Namely, we will make use of the following theorem.

**Theorem IV.1.** Let \( X = \{0, \ldots, K\} \) and define \( Y \) to be following set of single-point sets and intervals:
\[ Y = \{\{0\}, \{0, 1\}, \{1\}, \{1, 2\}, \ldots, \{K-1\}, \{K-1, K\}, \{K\} \]
(or equivalently, an indexing of this finite collection of intervals). Let \( (f_n) \) be the following family of functions: \( f_n \) maps \((x_1, x_2, \ldots, x_n)\) to the element of \( Y \) which contains the average \( \sum_i x_i/n \). Then, the family \( (f_n) \) is computable.

The proof of Theorem IV.1 is fairly involved and too long to be included in this extended abstract; however, we give an informal description of the algorithm for computing \( f_n \) in Section V. In this section, we show that Theorem IV.1 implies the computability of a large class of functions. We will say that a function \( h \) on the proportion set is computable if it corresponds to a proportion-based computable family \( (f_n) \). The level sets of \( h \) are defined as the sets \( L(y) = \{p \in D \mid h(p) = y\} \), for \( y \in Y \).

**Theorem IV.2 (Sufficient condition for computability).** Let \( h \) be a function from the proportion set \( D \) to \( Y \). Suppose that every level set \( L(y) \) can be written as a finite union,
\[ L(y) = \bigcup_k C_{i,k}, \]
where each \( C_{i,k} \) can in turn be written as a finite intersection of linear inequalities of the form
\[ \alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_K p_K \leq \alpha, \]
or
\[ \alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_K p_K < \alpha, \]
with rational coefficients \( \alpha, \alpha_1, \ldots, \alpha_K \). Then, \( h \) is computable.

**Proof:** Consider one such linear inequality. Let \( P \) be the set of indices \( i \) for which \( \alpha_i \geq 0 \). Since all coefficients are rational, we can clear the denominators and rewrite the inequality as
\[ \frac{\sum_{k \in P} \beta_k p_k}{n} - \sum_{i \in P^c} \beta_k p_k \leq \beta, \]
for nonnegative integers \( \beta_k \) and \( \beta \). Let \( \chi_k \) be the indicator function associated with initial value \( k \), i.e., \( \chi_k(i) = 1 \) if \( x_i = k \), and \( \chi_k(i) = 0 \) otherwise, so that \( p_k = \frac{1}{n} \sum \chi_k(i) \).

Then, (IV.1), becomes
\[ \frac{1}{n} \sum_{k \in P} \sum_{i=1}^n \beta_k \chi_k(i) + \sum_{k \in P^c} \beta_k (1-\chi_k(i)) \leq \beta + \sum_{k \in P^c} \beta_k, \]
or
\[ \frac{1}{n} \sum_{i=1}^n q_i \leq q^*, \]
where \( q_i = \sum_{k \in P} \beta_k \chi_k(i) + \sum_{k \in P^c} \beta_k (1-\chi_k(i)) \) and \( q^* = \beta + \sum_{k \in P^c} \beta_k \).

To determine if the latter inequality is satisfied, each node can compute \( q_i \) and \( q^* \), and then apply a distributed algorithm that computes \( \frac{1}{n} \sum_{i=1}^n q_i \), which is possible by virtue of Theorem IV.1. To check any finite collection of inequalities, the nodes can perform the computations for each inequality in parallel.

To compute \( h \), the nodes simply need to check which set \( L(y) \) the frequencies \( p_1, \ldots, p_K \) lie in, and this can be done by checking the inequalities defining each \( L(y) \). All of these computations can be accomplished with finite automata: indeed, we do nothing more than run finitely many copies of the automata provided by Theorem IV.1, one for each inequality.

Theorem IV.2 shows the computability of functions \( h \) whose level-sets can be defined by linear inequalities with rational coefficients. On the other hand, it is clear that not every function \( h \) can be computable. (This can be shown by a counting argument: there are uncountably many possible functions \( h \), but for the special case of bounded degree graphs, only countably possible algorithms.) Still, the next
lemma shows that the set of computable functions is rich enough, in the sense that such functions can approximate any measurable function.

We will call a set of the form $\prod_{k=1}^{K}(a_k, b_k)$, with every $a_k, b_k$ rational, a rational open box, where $\prod$ stands for Cartesian product. A function that can be written as a finite sum $\sum_i a_i \chi_{B_i}$, where the $B_i$ are rational open boxes and the $1_{B_i}$ are the associated indicator functions, will be referred to as a box function. Note that box functions are computable by Theorem IV.2.

**Corollary IV.3.** If every level set of a function $h : D \to Y$ on the proportion set is Lebesgue measurable, then, for every $\epsilon > 0$, there exists a computable box function $h_\epsilon : D \to Y$ such that the set $\{ h \neq h_\epsilon \}$ has measure at most $\epsilon$.

**Proof:** The proof relies on the following elementary result from measure theory. Given a Lebesgue measurable set $E \subseteq [0, 1]^K$ and some $\epsilon > 0$, there exists a set $E'$ which is a finite union of disjoint open boxes, and which satisfies

$$\mu((E - E') \cup (E' - E)) < \epsilon,$$

where $\mu$ is the Lebesgue measure. By a routine argument, these boxes can be taken to be rational. By applying this fact to the level sets of the function $h$ (assumed measurable), the function $h$ can be approximated by a box function $h_\epsilon$. Since box functions are computable, the result follows.

The following corollary states that quantizations of continuous functions are approximable.

**Corollary IV.4.** If a function $h : D \to [L, U] \subseteq \mathbb{R}$ is continuous, then for every $\epsilon > 0$ there exists a computable box function $h_\epsilon : D \to [L, U]$ such that $||h - h_\epsilon||_\infty < \epsilon$.

**Proof:** Since $D$ is compact, $f$ is uniformly continuous. One can therefore partition $D$ into a finite number of subsets $A_1, A_2, \ldots, A_q$, that can be described by linear inequalities with rational coefficients, so that $\max_{p \in A_j} h(p) - \min_{p \in A_j} h(p) < \epsilon$ holds for all $A_j$. The function $h_\epsilon$ is then built by assigning to each $A_j$ an appropriate value in $\{L, L + \epsilon, L + 2\epsilon, \ldots, U\}$.

Finally, we show that with infinite memory, it is possible to recover the exact frequencies, provided $n$ is unbounded, and the number of bits needed to represent $p_k$ is also unbounded.

**Theorem IV.5.** The vector $(p_1, \ldots, p_K)$ is computable with infinite memory.

**Proof:** We show that $p_1$ is computable exactly, which is sufficient to prove the theorem. Consider the following algorithm, parametrized by a positive integer $m$. The initial set $X_m$ will be $\{0, 1, \ldots, m\}$ and the output set $Y_m$ will be as in Theorem IV.1: $Y_m = \{\{0\}, \{0, 1\}, \{1\}, \{1, 2\}, \{2\}, \{2, 3\}, \ldots, \{m-1\}, \{m-1, m\}, \{m\}\}$. If $x_i = 1$, then node sends its initial value $x_{i,m}$ to $m$; else, the node sends its initial value $x_{i,m}$ to 0. The algorithm computes the function family $(f_n)$ which maps $X^n_m$ to the element of $Y_m$ containing $(1/n) \sum_{i=1}^{n} x_{i,m}$, which is possible, by Theorem IV.1. We will call this algorithm $Q_m$.

Let $y_m$ be its final output.

The nodes run the algorithms $Q_m$ for every positive integer value of $m$, in an interleaved manner. Namely, at each time step, a node runs one step of a particular algorithm $Q_m$, according to the following order:

$$Q_1, Q_1, Q_2, Q_1, Q_2, Q_3, Q_1, Q_2, Q_3, Q_4, Q_1, Q_2, \ldots$$

At each time $t$, let $m_i(t)$ be the smallest $m$ (if it exists) such that the output $y_{i,m_i(t)}$ of $Q_m$ at node $i$ is a singleton (not an interval). We identify this singleton with the numerical value of its single element, and we set $y_i(t) = y_{i,m_i(t)}(t)/m_i(t)$. If $m_i(t)$ is undefined, then $y_i(t)$ is set to some default value.

It follows from the definition of $Q_m$ and from Theorem IV.1 that there exists a time after which the outputs $y_{i,m}$ of the algorithms $Q_1, \ldots, Q_m$ do not change, and are the same for every node, denoted $y_m$. Moreover, at least one of these algorithms has an integer output $y_m$. Indeed observe that $Q_m$ computes $(1/n) \sum_{i=1}^{n} 1_{x_i=1} = \sum_{i=1}^{n} 1_{x_i=1}$, which is clearly an integer. In particular, $m_i(t)$ is eventually well-defined and bounded above by $n$. We conclude that there exists a time after which the output of our overall algorithm is fixed, shared by all nodes, and different from the default value.

We now argue that this value is indeed $p_1$. Let $m^*$ be the smallest $m$ for which the eventual output of $Q_m$ is a single integer $y_m$. Note that $y_{m^*}$ is the exact average of the $x_i, m^*$, i.e. $y_{m^*} = \frac{1}{n} \sum_{x_i=1}^{n} m^* 1_{x_i=1} = m^* p_1$. For large $t$, we have $y_i(t) = y_{i,m^*}(t)/m^* = p_1$.

Finally, it remains to argue that the algorithm described here can be implemented with a sequence of automata. All the above algorithm does is run a copy of all the automata implementing $Q_1, Q_2, \ldots$, with time-dependent transitions. This can be accomplished with an automaton whose state space is the countable set $\mathbb{N} \times \bigcup_{m=1}^{\infty} \prod_{i=1}^{m} \mathbb{Q}_i$, where $\mathbb{Q}_i$ is the state space of $Q_i$, and the set $\mathbb{N}$ of integers is used to keep track of time.

To illustrate the results of this section, let us consider again some examples.

(a) Majority testing between two options is equivalent to checking whether $p_1 \leq 1/2$, with alphabet $\{0, 1\}$, and is therefore computable.

(b) Majority testing when some nodes can “abstain” amounts to checking whether $p_1 - p_2 \geq 0$, with alphabet $\{0, 1, \text{abstain}\}$. This function family is computable.

(c) We can ask for the second most popular value out of four, for example. In this case, the sets $A_j$ can be decomposed into constituent sets defined by inequalities such as $p_2 \leq p_3 \leq p_4 \leq p_1$, each of which obviously has rational coefficients.

(d) For any subset $I, I'$ of $\{1, \ldots, K\}$, the indicator function of the set where $\sum_{i \in I} p_i \geq \sum_{i \in I'} p_i$ is computable. This is equivalent to checking whether more nodes have a value in $I$ than do in $I'$. 1586
(e) The indicator functions of the sets defined by $p_2^2 \leq 1/2$ and $p_1 \leq \pi/4$ are measurable, so they are approximable. We are unable to say whether they are computable.

(f) The indicator function of the set defined by $p_1 p_2 \leq 1/8$ is approximable, but we are unable to say whether it is computable.

V. A SKETCH OF THE PROOF OF THEOREM IV.1

In this section, we sketch an algorithm for computing the average of integer initial values, but omit the proof of correctness. We start with an important subroutine that tracks the maximum (over all nodes) of time-varying inputs at each node.

A. Distributed maximum tracking

Suppose that each node $i$ has a time-varying input $u_i(t)$ stored in memory at time $t$, belonging to a finite set of numbers $\mathcal{U}$. We assume that, for each $i$, the sequence $u_i(t)$ must eventually stop changing, i.e., there exists some $T'$ such that

$$u_i(t) = u_i(T'), \quad \text{for all } t \geq T'.$$

(However, the nodes need not be ever aware that $u_i(t)$ has reached its final value.) Our goal is to develop a distributed algorithm whose output eventually settles on the value $\max_i u_i(T')$. More precisely, each node $i$ is to maintain a number $M_i(t)$ which must satisfy the following constraint: for every connected graph and any allowed sequences $u_i(t)$, there exists some $T''$ with

$$M_i(t) = \max_{i=1,\ldots,n} u_i(t), \quad \text{for all } t \geq T''.$$

Moreover, node $i$ must also maintain a pointer $P_i(t)$ to a neighbor or to itself. We will use the notation $P_i^{(0)}(t) = P_{P_i(t)}(t)$, $P_i^{(1)}(t) = P_{P_i(t)}(t)$, etc. We require the following additional property, for all $t$ larger than $T''$: for each node $i$ there exists a node $j$ and a power $K$ such that for all $k \geq K$ we have $P_i^k = j$; moreover, $M_i(t) = u_j(t)$. In other words, by successively following the pointers $P_i(t)$, one can arrive at a node with the maximum value.

Theorem V.1. An algorithm satisfying the above conditions exists and can be implemented at each node with a finite automaton whose state can be stored using at most $C(\log |\mathcal{U}| + d(i))$ bits, for some absolute constant $C$.

We briefly summarize the algorithm guaranteed by this theorem. Each node $i$ initially sets $M_i(0) = u_i(0)$, $P_i(0) = i$. Nodes exchange their values $u_i(t)$ and forward the largest value they have seen; every node sets its estimated maximum $M_i(t)$ equal to that largest value, and sets its pointer $P_i$ to the node that forwarded that value to $i$. When some $u_i$ changes, the corresponding node sends out a reset message, which is then forwarded by all other nodes. The details of the algorithm and its analysis are somewhat involved because we need to make sure that the reset messages do not cycle forever.

B. The averaging algorithm

We continue with an intuitive description of the averaging algorithm. Imagine the initial integer values $x_i$ as represented by $x_i$ pebbles. Our algorithm attempts to exchange pebbles between nodes with unequal number of pebbles so that the overall distribution becomes more even. Eventually, either all nodes will have the same number of pebbles, or some will have a certain number and others just one more. We let $u_i(t)$ be the current number of pebbles at node $i$; in particular, $u_i(0) = x_i$. An important property of the algorithm is that the total number of pebbles is conserved.

To match nodes with unequal number of pebbles we use the maximum tracking algorithm of Section V-A. Recall that the algorithm provides nodes with pointers which attempt to track the location of the maximal values. When a node with $u_i$ pebbles comes to believe in this way that a node with at least $u_i + 2$ pebbles exists, it sends a request in the direction of the latter node to obtain one or more pebbles. This request follows a path to a node with a maximal number of pebbles, until the request either gets denied, or gets accepted by a node with at least $u_i + 2$ pebbles.

More formally, the algorithm uses two types of messages:

(a) (Request, $r$): This is a request for a transfer of value. Here, $r$ is an integer that represents the current number of pebbles at the emitter.

(b) (Accept, $w$): This corresponds to acceptance of a request, and subsequent transfer of $w$ pebbles to the requesting node. A request with a value $r = 0$ represents a request denial.

As part of the algorithm, the nodes run the maximum tracking algorithm of Section V-A, as well as a minimum tracking counterpart. In particular, each node $i$ has access to the variables $M_i(t)$ and $P_i(t)$ of the maximum tracking algorithm (recall that these are, respectively, the estimated maximum and a pointer to a neighbor). Furthermore, each node maintains three additional variables.

(a) “mode” ∈ {free, blocked}. Initially, the mode of every node is free. Nodes become blocked when they are handling requests.

(b) “Rin$_i(t)$”, “Rout$_i(t)$” are pointers to a neighbor of $i$ or to itself. They represent, respectively, the node from which $i$ has received a request, and the node to which $i$ has transmitted a request. Initially, Rin$_i(0) = Rout$_i(0) = 0$.

The algorithm is described in Figure 2.

A key step in the proof of Theorem IV.1 is the following proposition, whose proof is omitted.

Proposition V.1. There is a time $t'$ such that $u_i(t) = u_i(t')$, for all $i$ and $t \geq t'$. Moreover,

$$\sum_i u_i(t') = \sum_i u_i(0) = \sum_i x_i,$$

$$|u_i(t') - u_j(t')| \leq 1, \text{ for all } i, j.$$

We now conclude our sketch of the proof of Theorem IV.1. Let $u_i^*$ be the value that $u_i(t)$ settles on. It follows
from Proposition V.1 that if the average \( \bar{x} \) of the inputs \( x_i \) is integer, then \( u_i(t) = u_i^* = \bar{x} \) will eventually hold for every \( i \). If \( \bar{x} \) is not an integer, then some node will eventually have \( u_i^* = \lfloor \bar{x} \rfloor \) and others \( u_i^* = \lceil \bar{x} \rceil \). Using the maximum and minimum computation algorithm, nodes will eventually have a correct estimate of \( \max_i u_i^* \) and \( \min_i u_i^* \), because each \( u_i(t) \) converges to \( u_i^* \). This allows the nodes to determine if the average is exactly \( \bar{x} \) (integer average), or if it lies in \( (\lfloor \bar{x} \rfloor, \lceil \bar{x} \rceil + 1) \), or \( (\lceil \bar{x} \rceil - 1, \lceil \bar{x} \rceil) \), which is the property asserted by Theorem IV.1.

REFERENCES
