arXiv:0907.2949v2 [cs.DC] 28 Jul 2009

Distributed anonymous function computation in information fusion and multiagent systems*

Julien M. Hendrickx, Alex Olshevsky, John N. Tsitsiklis

July 28, 2009

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.

1 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 authors are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, jm_hend@mit.edu, alex_o@mit.edu, jnt@mit.edu. This research was supported by the National Science Foundation under grant ECCS-0701623, and postdoctoral fellowships from the F.R.S.-FNRS (Belgian Fund for Scientific Research) and the B.A.E.F. (Belgian American Education Foundation).

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].

1.1 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.

1.2 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 be 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, but 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.

2 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 numbers* in the set $\{1, 2, \ldots, \text{degree}(i)\}$; (iii) a family of finite automata $(A_d)_{d=1,2,3,\ldots}$. (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, $\log |Z_d| \leq Cd$, 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 $S_i(t) = [x_i, y_i(t), z_i(t), m_{i,1}(t), \ldots, m_{i,d(i)}(t)]$ to denote the state of the automaton implemented by agent i at round t. Let $j_1, \ldots, j_{d(i)}$ be an enumeration of the neighbors of i, and let p_k be the port number of the link (j_k, i). The evolution of the system is then described by

$$\left[x_i, z_i(t+1), y_i(t+1); m_{i,1}(t+1), \ldots, m_{i,d(i)}(t+1)\right] = A_{d(i)}\left[x, z_i(t), y_i(t); m_{j_1,p_1}(t), \ldots, m_{j_d,p_d}(t)\right].$$

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 $y^* \in Y$ is the *final output* of this dynamical system if there is a time t' such that $y_i(t) = y^*$ for every i and $t \geq t'$.

Consider now a family of functions $(f_n)_{n \in \mathcal{N}} : X^n \to Y$. We say that such a family is *computable* if there exists a family of automata $(A_d)_{d=1,2,\ldots}$ such that for any n, for any connected graph $G = (V, E)$ with n nodes, any port labelling, and any set of initial conditions $x_1, \ldots, x_n$, the final output of the above system is always $f_n(x_1, \ldots, x_n)$.

In some results, we will also refer to function families $(f_n)_{n \in \mathcal{N}}$ *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?

3 Necessary condition for computability

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

Lemma 3.1. *Suppose that $G = (\{1, \ldots, n\}, E)$ and $G' = (\{1, \ldots, n\}, E')$ are isomorphic, that is, there exists a permutation π 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 $x_1, \ldots, x_n$ with the graph G is the same as the state $S_{\pi(i)}(t)$ resulting from the initial values $x_{\pi(1)}, \ldots, x_{\pi(n)}$ with the graph G'.*

Proposition 3.1. *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 π_{ij} be permutation that swaps i with j; with a slight abuse of notation, we also denote by π_{ij} the mapping from X^n to X^n that swaps the ith and jth 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 3.1 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 π_{ij} generate the group of permutations, permutation invariance follows. □

Let $x \in X^n$. We will denote by x^2 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 3.2. *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 ith agent counterclockwise begins with the ith element of x; and consider a ring of size km where the agents $i, i+m, i+2m, \ldots$ (counterclockwise) begin with the ith 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 $x_i = i$.

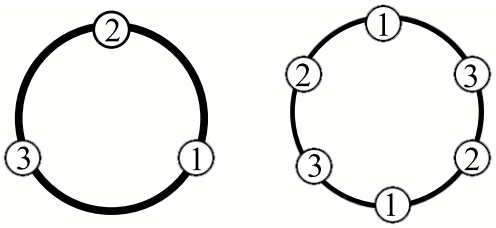

Figure 1: Example of two situations that are indistinguishable by the nodes.

Initially, the state of node i in the first ring is exactly the same as the state of the nodes $j = i, i+m, i+2m, \ldots$ in the second ring. We show by induction that this property must hold at all times t. (To keep notation simple, we assume, without any real loss of generality, that $i \neq 1$ and $i \neq m$.)

Indeed, suppose this property holds up to time t. At time t, node i in the first ring receives a message from node $i-1$ and a message from node $i+1$; and in the second ring, node j satisfying $j \pmod m = i$ receives one message from $j-1$ and $j+1$. Since $j-1 \pmod m = i-1$ and $j+1 \pmod m = i+1$, the states of $j-1$ and $i-1$ are identical at time t, and similarly for and $j+1$ and $i+1$. Thus the messages received by i (in the first ring) and j (in the second ring) at time t are identical. Since i and j were in the same state at time t, they must be in the same state at time $t+1$. This proves that they are in the same state forever.

It follows that $y_i(t) = y_j(t)$ for all t, whenever $j \pmod m = i$, and therefore $f_m(x) = f_{km}(x^k)$. □

We can now prove our main negative result, stating that if a family of functions is computable, then the value of the function only depends on the frequencies of the different possible initial values. We define $D = \{(p_1, \ldots, p_K) \in [0,1]^K : \sum_{k=1}^K p_k = 1\}$, which we call the *proportion set*. We say that a function $h : D \to Y$ corresponds to a family $(f_n : X^n \to Y)$ if for every $x \in X^n$,

$$f(x_1, \ldots, x_n) = h\left(p_1(x_1, \ldots, x_n), \ldots, p_K(x_1, \ldots, x_n)\right),$$

where

$$p_k(x_1, \ldots, x_n) = |\{i \mid x_i = k, \}|/n,$$

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 3.1. *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 are $n\hat{p}_k$ and $m\hat{p}_k$, respectively. Observe that for any $k \in X$, the vectors x^m 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 3.1 implies that $f_{nm}(x^m) = f_{nm}(y^n)$. It then follows from Proposition 3.2 that $f_{nm}(x^m) = f_n(x)$ and $f_{mn}(y^n) = f_m(y)$, so that $f_n(x) = f_m(y)$. 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 $\sum_{i=1}^{n} x_i \pmod k$ 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 $|i : \{x_i = 0\}| = 1$, is not computable.
(d) An aggregate difference functions such as $\sum_{i<j} |x_i - x_j|$ is not computable, even calculated modulo k.

4 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 4.1. *Let $X = \{0, \dots, K\}$ and define Y to be following set of single-point sets and intervals:*

$$Y = \{\{0\}, (0,1), \{1\}, (1,2), \dots, \{K-1\}, (K-1, K), \{K\}\}$$

(or equivalently, an indexing of this finite collection of intervals). Let (f_n) be the following family of functions: f_n maps $(x_1, x_2, \dots, x_n)$ 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 4.1 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 5. In this section, we show that Theorem 4.1 implies the computability of a large class of functions. We will say that a function h on the proportion set is *computable* if it 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 4.2 (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 $\alpha, \alpha_1, \dots, \alpha_K$. 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

$$\sum_{k \in P} \beta_k p_k - \sum_{i \in P^c} \beta_k p_k \leq \beta, \tag{4.1}$$

for nonnegative integers β_k and β. Let χ_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 $p_k = \frac{1}{n} \sum_i \chi_k(i)$. Then, (4.1), becomes

$$\frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k \in P} \beta_k \chi_k(i) + \sum_{k \in P^c} \beta_k (1 - \chi_k(i)) \right) \leq \beta + \sum_{k \in P^c} \beta_k,$$

or

$$\frac{1}{n} \sum_i q_i \leq q^*,$$

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 4.1. 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 $p_1, \dots, p_K$ 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 4.1, one for each inequality. ∎

Theorem 4.2 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 $\sum_i a_i 1_{B_i}$, 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 4.2.

Corollary 4.3. *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 ϵ.*

Proof. The proof relies on the following elementary result from measure theory. Given a Lebesgue measurable set $E \in [0,1]^K$ 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 μ 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_ϵ. Since box functions are computable, the result follows. □

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

Corollary 4.4. *If a function $h : D \to [L, U] \subseteq \Re$ is continuous, then for every $\epsilon > 0$ there exists a computable 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_ϵ 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 p_k. (Note that this is impossible with finite memory, because n is unbounded, and the number of bits needed to represent p_k is also unbounded.)

Theorem 4.5. *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 4.1: $Y_m = \{\{0\}, (0,1), \{1\}, (1,2), \{2\}, (2,3), \ldots, \{m-1\}, (m-1, m), \{m\}\}$. If $x_i = 1$, then node sets its initial value $x_{i,m}$ to m; else, the node sets its initial value $x_{i,m}$ to 0. The algorithm computes the function family (f_n) which maps X_m^n to the element of Y_m containing $(1/n)\sum_{i=1}^{n} x_{i,m}$, which is possible, by Theorem 4.1. 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 $y_{i,m}(t)$ 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 4.1 that there exists a time after which the outputs $y_{i,m}$ of the algorithms $Q_1, \ldots, Q_n$ 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_n computes $(1/n)\sum_{i=1}^{n} n 1_{x_i=1} = \sum_{i=1}^{n} 1_{x_i=1}$, 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. $y_{m^*} = \frac{1}{n}\sum_{i=1}^{n} m^* 1_{x_i=1} = m^* p_1$. 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 $\mathcal{N} \times \cup_{m=1}^{\infty} \prod_{i=1}^{m} \mathcal{Q}_i$, where $\mathcal{Q}_i$ is the state space of Q_i, and the set $\mathcal{N}$ 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_i 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 subsets 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'.

(e) The indicator functions of the sets defined by $p_1^2 \leq 1/2$ 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.

5 A sketch of the proof of Theorem 4.1

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.

5.1 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., that there exists some T' such that

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

(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

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

Moreover, node i must also maintain a pointer $P_i(t)$ to a neighbor or to itself. We will use the notation $P_i^2(t) = P_{P_i(t)}(t)$, $P_i^3(t) = P_{P_i^2(t)}(t)$, 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 5.1. *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.

5.2 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 5.1. 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 $w = 0$ represents a request denial.

As part of the algorithm, the nodes run the maximum tracking algorithm of Section 5.1, 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"$\in$ {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, $Rin_i(0) = Rout_i(0) = \emptyset$.

The algorithm is described in Figure 2.

A key step in the proof of Theorem 4.1 is the following proposition, whose proof is omitted.

Proposition 5.1. *There is a time t' such that $u_i(t) = u_i(t')$, for all i and $t \geq t'$. Moreover,*

$$\begin{aligned} \sum_i u_i(t') &= \sum_i u_i(0) = \sum_i x_i, \\ |u_i(t') - u_j(t')| &\leq 1, \text{ for all } i, j. \end{aligned}$$

We now conclude our sketch of the proof of Theorem 4.1. Let u_i^* be the value that $u_i(t)$ settles on. It follows from Proposition 5.1 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 $(\bar{u}_i^*, \bar{u}_i^* + 1)$, or $(\bar{u}_i^* - 1, \bar{u}_i^*)$, which is the property asserted by Theorem 4.1.

References

[1] "Local and global properties in networks of processors," D. Angluin, *Proceedings of the twelfth annual ACM symposium on Theory of computing*, 1980.

[2] "An introduction to population protocols," J. Aspnes and E. Ruppert, *Bulletin of the European Association for Theoretical Computer Science,* 93:98117, October 2007.

[3] "Distributed Average Consensus using Probabilistic Quantization," T.C. Aysal, M. Coates, M. Rabbat, *14th IEEE/SP Workshop on Statistical Signal Processing,* 2007.

[4] Y. Afek and Y. Matias, "Elections in anonymous networks," *Information and Computation,* 113:2, pp. 113-330, 1994.

[5] O. Ayaso, D. Shah, M. Dahleh, "Counting bits for distributed function computation," *Proceedings of the IEEE International Symposium on Information Theory,* 2008.

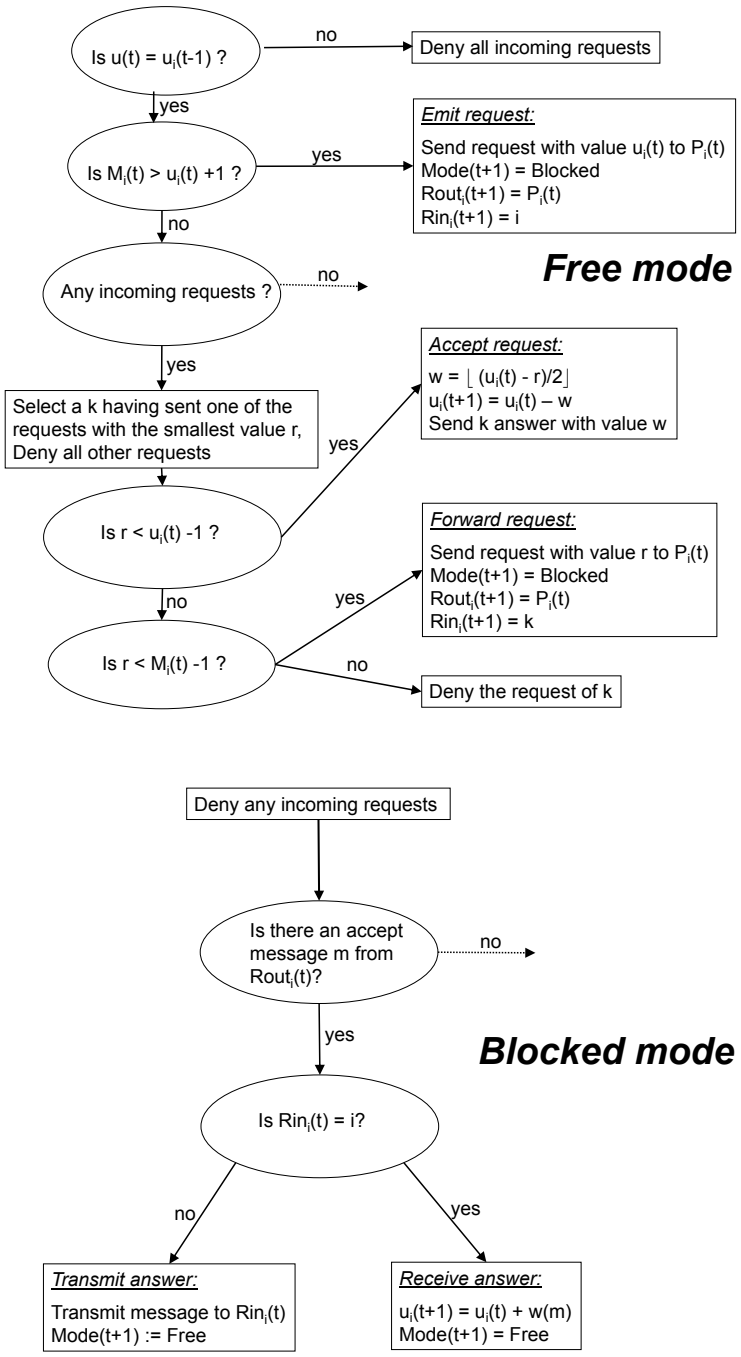

Figure 2: Representation of the average computation algorithm

[6] H. Attiya, M. Snir, M.K. Warmuth, "Computing on an anonymous ring," *Journal of the ACM,* Vol. 35, No. 4, pp. 845 - 875, 1988.

[7] F. Bénézit, P. Thiran, M. Vetterli"Interval consensus: from quantized gossip to voting," , *Proceedings of ICASSP 08.*

[8] R. Carli, F. Bullo, "Quantized Coordination Algorithms for Rendezvous and Deployment, " preprint.

[9] P. Frasca, R. Carli, F. Fagnani, S. Zampieri, "Average consensus on networks with quantized communication," preprint.

[10] F. Fich, E. Ruppert, "Hundreds of impossibility results for distributed computing," *Distributed Computing,* Vol. 16, pp. 121-163, 2003.

[11] A. Giridhar, P.R. Kumar, "Computing and communicating functions over sensor networks," *IEEE Journal on on Selected Areas in Communications,* Vol. 23, No. 4, pp. 755- 764, 2005.

[12] P. Gacs, G.L. Kurdyumov, L.A. Levin, "One-dimensional uniform arrays that wash out finite islands," *Problemy Peredachi Informatsii,* 1978.

[13] A.G. Greenberg, P. Flajolet, R. Lander, "Estimating the multiplicity of conflicts to speed their resolution in multiple access channels," Journal of the ACM, Vol 34, No. 2, 1987.

[14] Y. Hassin and D. Peleg, "Distributed Probabilistic Polling and Applications to Proportionate Agreement," *Information and Computation* 171, (2001), 248-268.

[15] A. Jadbabaie, J. Lin, A.S. Morse, " Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control,* Vol. 48, No. 6, pp. 988-1001, 2003.

[16] A. Kashyap, T. Basar, R. Srikant, "Quantized consensus," *Automatica,* Vol. 43, No. 7, pp. 1192-1203, 2007.

[17] E. Kranakis, D. Krizanc, J. van den Berg, "Computing boolean functions on anonymous networks," *Proceedings of the 17th International Colloquium on Automata, Languages and Programming,* Warwick University, England, July 1620, 1990.

[18] N. Khude, A. Kumar, A. Karnik, "Time and energy complexity of distributed computation in wireless sensor networks," *Proceedings of INFOCOM 05.*

[19] N. Katenka, E. Levina, and G. Michailidis, "Local Vote Decision Fusion for Target Detection in Wireless Sensor Networks," *IEEE Transactions on Signal Processing,* 56(1):329-338, 2008.

[20] S. Kar, J.M Moura, "Distributed average consensus in sensor networks with quantized inter-sensor communication, " *IEEE International Conference on Acoustics, Speech and Signal Processing,* 2008.

[21] M. Land, R.K. Belew, "No perfect two-state cellular automaton for density classification exists, " *Physical Review Letters,* vol 74, no. 25, pp. 5148-5150, Jun 1995.

[22] Y. Lei, R. Srikant, G.E. Dullerud, "Distributed Symmetric Function Computation in Noisy Wireless Sensor Networks," *IEEE Transactions on Information Theory,* Vol. 53, No. 12, pp. 4826-4833, 2007.

[23] L. Liss, Y. Birk, R. Wolff, and A. Schuster, "A Local Algorithm for Ad Hoc Majority Voting Via Charge Fusion," *Proceedings of DISC'04*, Amsterdam, the Netherlands, October, 2004

[24] S. Martinez, F. Bullo, J. Cortes, E. Frazzoli, "On synchronous robotic networks - part I: models, tasks, complexity," *IEEE Transactions on Robotics and Automation,* Vol. 52, No. 12, pp. 2199 - 2213, 2007.

[25] S. Mukherjeea, H. Kargupta, "Distributed probabilistic inferencing in sensor networks using variational approximation," *Journal of Parallel and Distributed Computing,* Volume 68, Issue 1, Pages 78-92, 2008.

[26] S. Moran, M.K. Warmuth, "Gap Theorems for Distributed Computation," *SIAM Journal of Computing* Vol. 22, No. 2, pp. 379-394 (1993).

[27] A. Nedic, A. Olshevsky, A. Ozdaglar, J.N. Tsitsiklis, "On distributed averaging algorithms and quantization effects," preprint, to be published in *Proceedings of 47th IEEE Conference on Decision and Control,* 2008.

[28] D. Peleg, "Local majorities, coalitions and monopolies in graphs: a review, " *Theoretical Computer Science,* Vol. 282, No. 2, pp. 231-237, 2002.

[29] E. Perron, D. Vasuvedan, M. Vojnovic, "Using Three States for Binary Consensus on Complete Graphs," preprint.

[30] L. Xiao and S. Boyd, "Fast Linear Iterations for Distributed Averaging," *Systems and Control Letters,* 53:65-78, 2004.

[31] M. Yamashita and T. Kameda, "Computing on an anonymous network," *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing,* 1988.