Fundamental limitations for anonymous distributed systems with broadcast communications

Julien M. Hendrickx and John N. Tsitsiklis

***Abstract*— We consider deterministic anonymous distributed systems with broadcast communications where each node has some initial value, and the goal is to compute a function of all these values. We show that only a very restricted set of functions can be computed if the nodes do not know (and cannot use) the number of their out-neighbors. Our results remain valid even if nodes know the precise structure of the network but do not know where they lie within the structure. They also remain valid if nodes know their out-degree up to an uncertainty of 1. These results are a variation of those obtained by Boldi and Vigna (1997) for a weaker computation model.**

As a consequence, computing more complex functions in the context of broadcast communications requires the explicit or implicit knowledge or use of either (a) the out-degree of each node, (b) global node identifiers, (c) randomization, or (d) asynchronous updates with specific properties.

I. Introduction

We consider a model of anonymous distributed computation where nodes communicate by broadcasting messages, they have thus to send the same message to all those with which communication is possible.

We assume that nodes can distinguish the emitters of the different messages they receive (their in-neighbors), but do not know how many nodes receive the messages they broadcast (their out-neighbors). We do not assume the communication links to be symmetric, as various practical constraints (power, geometry etc.) may make it possible for a node j to receive messages from i but impossible for i to receive messages from j. Nodes act synchronously and can perform arbitrary but deterministic internal computations.

Each node holds an initial input value x_i and the goal is for all the nodes to collaboratively compute a function of these input values x_i. We will explore the fundamental limitations on the functions that can be computed in our model.

Different recent works in the control and communication literature concern the design of algorithms to compute functions on systems with (possibly) asymmetric broadcast communications. The problems studied include, for example, evaluating the average of the node values [5], [6], [10], [14], distributed optimization problems, in which each node holds a function and the goal is to find a minimizer of the sum of these functions [1], [12], [13], [15], [16], or the distributed computation of a vector satisfying a set of constraints held by the nodes [11].

In many of these works, it is assumed that *nodes know their out-degree, i.e., the number of nodes receiving the messages they broadcast*, see, e.g., [1], [12], [13], [16], or [6] in a slightly different context. Alternatively, nodes are sometimes provided with equivalent information, such as a set of weights A_{ij} whose sum over their out-neighbors and themselves equals 1; see, e.g. [5], [10], [15]. We note that such information cannot be obtained by the nodes directly if they do not know their out-degree, but would have to be externally supplied.

We believe that having nodes knowing the out-degrees is a nontrivial assumption. Indeed, consider a directed link (i, j), meaning that the messages broadcast by i reach j but j cannot communicate directly with i. How would it be possible for i to know a priori that its messages are received by j? One option is to provide this information to i when the network is established. This requires an external intervention not necessarily convenient in the context of self-organized systems. Moreover, this information may eventually become outdated, as certain links may disappear or appear if the network is deployed over a long period of time. External interventions would then be needed again to re-configure the network. Alternatively, some systems may allow nodes to occasionally use more transmission power, allowing j to inform i every once in a while that it receives its messages, but this is not necessarily always possible. Another option is to have a first phase of the algorithm that computes the out-degrees, using for example identifiers or random numbers. One could then argue that this first phase is a part of the algorithm, and that the nodes initially did not know their out-degrees. Finally, even when node out-degrees can be detected or supplied, knowing whether their knowledge is necessary is of interest to an algorithm designer, especially if this involves an additional implementation cost.

Our main message is that the systems we consider, and in which nodes do not know their out-degrees, can only compute the restricted set of *order- and multiplicity-independent functions*: functions depending only on the set of values held by nodes in the network and not on the number of nodes holding each value. Those include, for example, the minimum and the maximum, but not the average.

This fundamental limitation had been proved by Boldi and Vigna [3, Theorem 15] for systems where nodes are not only ignorant of their out-degrees but are also unable to

Julien Hendrickx is with the ICTEAM institute, Université catholique de Louvain, Louvain-la-Neuve, Belgium. julien.hendrickx@uclouvain.be, John Tsitsiklis is with LIDS, Massachusetts Institute of Technology, Cambridge, MA 02139, jnt@mit.edu.

This work was supported by the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Program, initiated by the Belgian Science Policy Office.

distinguish between the emitters of the different messages they receive. Their proof relies on the notion of graph fibration which plays an essential role in different works on distributed computation [4], and is also related to the notion of a "view" of a network by a node, introduced in [18]. It turns out their proof can be directly extended to the systems we consider. In addition, in their work classifying various weak models of distributed computation, Hella *et al.* [8] argue that in the absence of restrictions on node memory and messages, models where nodes are able to distinguish the emitters of the broadcasted messages that they receive are equivalent to models where they are unable to do so. Note that the latter work considers algorithms that compute properties of the network or identify certain structures within the network, as opposed to computing functions of inputs held by the nodes, but their argument remains valid in the context of function computation.

In this work, we present an alternative proof of this fundamental limitation, also showing that it applies even if nodes know their out-degrees up to an uncertainty of 1. (By contrast, the proof in of Boldi and Vigna relies on arbitrarily large uncertainties on the degrees). We also consider the problem of approximately computing a function, and show that this cannot be done with any meaningful accuracy guarantee, even if nodes know the network and their out-degree up to an uncertainty of 1. (We note that our negative results actually apply to a somewhat weaker requirement than that in the previous literatire, in that we only require eventual convergence to a desired answer, as in average consensus, optimization, etc.) Finally, we connect our negative results to various problems that have attracted some attention in the recent control and optimization literature, and explore how they relate to models where a network is assumed to be represented by a (often stochastic) matrix A; see, e.g., [5], [7], [10], [15].

II. Model

Network: The network is modeled as a fixed directed graph $G{=}(V, E)$ with an incoming port numbering. The presence of a directed edge $(i, j) \in E$ means that node $j \in V$ receives the messages that node $i \in V$ broadcasts. The incoming port-numbering assigns a distinct port number in $\{1, 2, \dots, d_i^-\}$ to each incoming edge of every node i, where d_i^- denotes the out-degree of node i, and thus the number of nodes from which it can receive messages. These port numbers represent the ability of the nodes to distinguish the emitters of the different messages they receive. We do not consider self-loops, and assume $n > 2$ to avoid trivial cases.

It is important to note that nodes do not know the identifiers i that we use in our analysis, and they cannot use them in their computations.

Node variables and messages: To each node i are associated four variables $x_i \in X$, $y_i \in Y$, $z_i \in Z$ and $b_i \in B$. The *input* $x_i \in X$ is constant over time and represents an observation made by the node or its initial information. The output $y_i(t)$ represents the estimated answer of node i at time t, and can be updated at each time. In addition, nodes have some internal memory state $z_i(t)$ to help in performing their computation, and may broadcast at each time a (unique) message $b_i(t)$.

Algorithms: An algorithm is a family of update functions $(A_d)_{d=1,2,\dots}$, where the index d represents the in-degree of the node, and thus the number of messages that it receives. At each time $t \geq 0$, every node i with in-degree d_i^- and in-neighbors $j_1, j_2, \dots, j_{d_i^-}$ with corresponding port-numbers $1, 2, \dots, d_i^-$, updates its variables according to

$$\begin{aligned}&(y_i(t+1), z_i(t+1), b_i(t+1)) \\ &= A_{d_i^-}\left(x_i(t); y_i(t); z_i(t); b_{j_1}(t), \dots, b_{j_{d_i^-}}(t)\right).\end{aligned} \tag{1}$$

In other words, based on all its internal variables and on the messages broadcast by its in-neighbors, the node computes its new estimated answer y_i, its new internal state z_i, and its new broadcast message b_i. This computation may depend on the node in-degree, but not on its out-degree. For the initialization, we assume that the sets Y, Z, B contain a special default element $\varnothing$, and that $y_i(0) = z_i(0) = b_i(0) = \varnothing$. The execution of the algorithm and the evolution of the variables depends thus only on the inputs $x_1, \dots, x_n$ (where n is the number of nodes). We will therefore say that the execution or the system starts on $x = (x_1, x_2, \dots, x_n)$.

Function computation: In order to also include systems asymptotically converging to a desired value, we assume that the set Y of estimated answers is endowed with a distance. This is not a loss of generality, as one can always use the trivial distance function $\operatorname{dist}(x, y) = 0$ if $x = y$, and $\operatorname{dist}(x, y) = 1$ if $x \neq y$. We say that the *system converges to the output* $y^* \in Y$ if $\lim_{t\to\infty} \operatorname{dist}(y_i(t), y^*) = 0$, for all i. In the particular cases where Y is finite or where the trivial distance function is used, convergence to the output y^* is equivalent to the existence of a time t^* after which $y_i(t) = y^*$ for all i and $t \geq t^*$.

The objective of the computation is to evaluate a certain function, which will naturally also depend on the number n of nodes. Thus, we consider a family of functions $(f_n)_{n=1,2,\dots} : X^n \to Y : (x_1, x_2, \dots, x_n) \to f_n(x_1, x_2, \dots, x_n)$ of the inputs. Such a family of functions associates thus one value in the output set Y to any sequence of values of the input set. We say that an algorithm $(A_d)_{d=1,2,\dots}$ *(asymptotically) computes* the family of functions $(f_n)_{n=1,2,\dots}$ if, for any strongly connected network on n nodes (and associated incoming port-numbering), and any values $(x_1, x_2, \dots, x_n)$, the system obtained by executing that algorithm converges to $y^* = f_n(x_1, x_2, \dots, x_n)$ when starting on $(x_1, x_2, \dots, x_n)$. A family of functions is computable if there exists an algorithm that computes it.

Since our goal is to establish fundamental limitations for a model as general as possible, we do not impose any

restriction on the nature of the sets of internal memory states Z and messages B, on their finite or infinite cardinality and dimensions, nor on the update functions A_d defining the algorithm.

III. Fundamental limitations

The first lemma reflects the arbitrary character of the node identifiers we use when analyzing the system: since we can change these identifiers without affecting the "real" system, the functions computed and every property of the system should be invariant under a permutation of these identifiers.

Lemma 1: If a family $(f_n)_{n=1,2\dots}$ is computable, then each f_n is invariant under permutation of its arguments.

Proof: Many variations of this result appear in the literature. We refer for example to [9] for a proof in a similar context. ∎

To take advantage of this invariance under permutation, we define the *list of occurrences* $(x_A^{n_A}, x_B^{n_B}, \dots)$ associated to a vector $x = (x_1, x_2, \dots, x_n)$ as the sorted list of all values $x_A, x_B, \dots$ appearing in x, where each value also has an associated superscript that indicates the number of times it appears in x, which we call its "number of occurrences." For example, the list of occurrences corresponding to the vector (p, u, p, r) is (p^2, r, u). Observe that $n = \sum_\alpha n_\alpha$. Note that this notation implicitly assumes X to have a total order, but one can verify that our results also hold without this assumption. Lemma 1 implies that if a family $(f_n)_{n=1,2,\dots}$ is computable, then there exists an associated function $\tilde{f}$ that takes as input a list of occurrences of elements of X, and which satisfies $f(x_1, x_2, \dots, x_n) = \tilde{f}(x_A^{n_A}, x_B^{n_B}, \dots)$, for any $x = (x_1, x_2, \dots, x_n)$ and associated list of occurrences $(x_A^{n_A}, x_B^{n_B}, \dots)$.

The next lemma is central to our argument. It shows that when $n \geq 4$, the value of a computable function does not change when the number of occurrences of a value is decreased by 1 provided that it remains positive.

Lemma 2: Let $(f_n)_{n=1,2\dots}$ be a computable family of functions and let $\tilde{f}$ be its associated representation. Let $(x_A^{n_A}, x_B^{n_B}, \dots)$ be a list of occurrences such that $n = \sum_\alpha n_\alpha \geq 4$. For any value x_K in the list for which $n_K \geq 2$, there holds

$$\tilde{f}(x_A^{n_A}, \dots, x_K^{n_K}, \dots) = \tilde{f}(x_A^{n_A}, \dots, x_K^{n_K - 1}, \dots) \qquad (2)$$

Proof: Take an arbitrary list of occurrences $(x_A^{n_A}, \dots, x_K^{n_K}, \dots)$ for which $\sum_\alpha n_\alpha \geq 4$ and $n_K \geq 2$. We analyze the executions of an algorithm computing $(f_n)_{n=1,2\dots}$ on the two networks G and G' represented in Fig. 1 for some inputs x and x' corresponding to $(x_A^{n_A}, \dots, x_K^{n_K}, \dots)$ and $(x_A^{n_A}, \dots, x_K^{n_K - 1}, \dots)$ respectively, and show they produce the same result.

For the network G, we consider an input vector $x \in X^n$ for which $x_1 = x_n = x_K$, while the remaining x_i are arbitrary but consistent with the list of occurrences. This is always possible since $n_K \geq 2$. The input vector $x' \in X^{n-1}$ for G' is then obtained from x by just removing x_n. Its corresponding list of occurrences is thus $(x_A^{n_A}, \dots, x_K^{n_K - 1}, \dots)$.

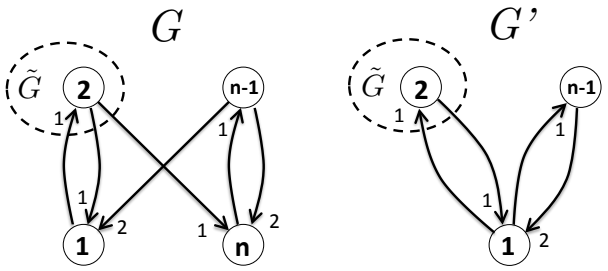

Fig. 1. Networks on n and $n-1$ nodes used to establish Lemma 2. The labels next to each arrow correspond to the port numbers. Here, $\tilde{G}$ is an arbitrary strongly connected network. When node 1 and n have the same initial input in G, it is impossible for nodes to know whether they are in network G or G'.

We denote by b'_i, y'_i and z'_i the broadcast messages, estimated answer, and internal state of node i in the network G'. We now show by induction that for all $t \geq 0$, there holds

$$\begin{aligned} [b, y, z]_i(t) &= [b', y', z']_i(t) \text{ for } i = 1, \dots, n-1 \\ [b, y, z]_n(t) &= [b, y, z]_1(t) \end{aligned} \qquad (3)$$

The case $t = 0$ follows directly from the initialization of all these variables at $\varnothing$. Suppose now that the equality holds at some time $t \geq 0$ and let us prove it holds at time $t+1$.

Nodes $1, 2 \dots, n-2$ have the same in-neighbors and the same port numbers in G and G', and these in-neighbors have sent the same messages at time t by the induction hypothesis. Thus, nodes $1, \dots, n-2$ receive the same messages in G and G'. Node $n-1$ has n as in-neighbor in G and 1 as in-neighbor in G', with port number 1 in both cases. The induction hypothesis (3) implies that $b_n(t) = b_1(t) = b'_1(t)$, so that node $n-1$ also receives the same message in G and G'. By construction, there also holds $x_i = x'_i$ for $i = 1, \dots, n-1$. All this together with the induction hypothesis (3) shows that every node $i \in \{1, \dots, n-1\}$ is exactly in the same situation at time t in G and G' and computes, using (1), the same new values: $[b_i, y_i, z_i](t+1) = [b'_i, y'_i, z'_i](t+1)$.

Observe now that node n and node 1 both have as in-neighbors in G node 2 with port number 1 and node $n-1$ with port number 2, so that they always receive the same messages. There holds moreover $x_1 = x_n$ by construction and $[b_n, y_n, z_n](t) = [b_1, y_1, z_1](t)$ by the induction hypothesis (3). Therefore, nodes 1 and n compute the same new values, using (1), so that $[b_n, y_n, z_n](t+1) = [b_1, y_1, z_1](t+1)$, which establishes condition (3) for $t+1$, and thus for all $t \geq 0$.

Since the algorithm computes $(f_n)_{n=1,2\dots}$, every estimated answer $y_i(t)$ in G converges to $f(x) = \tilde{f}(x_A^{n_A}, \dots, x_K^{n_K}, \dots)$ and every estimated answer $y'_i(t)$ in G' converges to $f(x') = \tilde{f}(x_A^{n_A}, \dots, x_K^{n_K - 1}, \dots)$. Because $y_i(t) = y'_i(t)$ holds for all t and $i = 1, \dots, n-1$, this establishes the equality in (2). ∎

We now need a variation of Lemma 2 to treat some special cases where the vector x contains only one or two values.

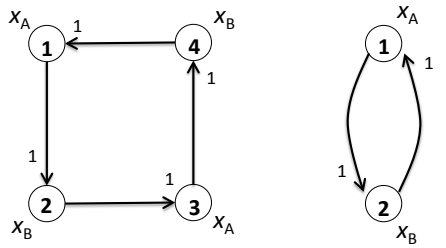

Fig. 2. Networks and input values used to establish Lemma 3. Any algorithm will compute the same value in both networks, (a) and (b).

Lemma 3: Let $(f_n)_{n=1,2\dots}$ be a computable family of functions and $\tilde{f}$ be its associated representation. For any values $x_A, x_B \in X$, there holds
(i) $\tilde{f}(x_A^1, x_B^1) = \tilde{f}(x_A^2, x_B^2)$,
(ii) $\tilde{f}(x_A^2) = \tilde{f}(x_A^4)$.

Proof: The result is a instance of a relatively standard argument in distributed computation, see for example [9, Lemma 3.3]. It can be proved by induction, similar to the proof of Lemma 2, using the two networks and inputs shown in Fig. 2. ∎

We say that a family of function $(f_n)_{n=1,2,\dots}$ is *order- and multiplicity-independent* if its value is entirely determined by the set of values appearing in the input vector x (as opposed to the order in which they appear and the number of their occurrences). For example the minimum and maximum functions are order- and multiplicity-independent, but the average is not. Formally, the family of functions is order- and multiplicity-independent if there exists a function $f^s : 2^X \to Y$ such that $f_n(x_1, x_2, \dots) = f^s(\{x_1, x_2, \dots\}) = f^s(\{\xi \in X : \exists i : \xi = x_i\})$.

Theorem 1: If a family of function is computable according to the model described in Section II then it is order- and multiplicity-independent, i.e. its value only depends on the *set* of input values $\{x_1, x_2, \dots\} = \{\xi \in X : \exists i : \xi = x_i\}$.

Proof: We show that for any computable function, there holds $\tilde{f}(x_A^{n_A}, x_B^{n_B}, x_C^{n_C}, \dots) = \tilde{f}(x_A^1, x_B^1, x_C^1, \dots)$ (or $\tilde{f}(x_A^2)$ in the case where the list of occurrence contains a single value). We consider three different cases.

Suppose first that a list of occurrence $(x_A^{n_A}, x_B^{n_B}, x_C^{n_C}, \dots)$ contains at least three distinct values. A repeated application of Lemma 2 shows then $\tilde{f}(x_A^{n_A}, x_B^{n_B}, x_C^{n_C}, \dots) = \tilde{f}(x_A^1, x_B^1, x_C^1, \dots)$. Indeed, as long as $n_K > 1$ for some x_K, there necessarily holds $\sum_\alpha n_\alpha > 3$ because the other values occur at least once, and we can apply Lemma 2 to show that decreasing the number of occurrences of x_K by 1 does not change the value of the function.

The case of a list of occurrences with two values requires some additional care due to the condition $\sum_\alpha n_\alpha > 3$ in Lemma 2. Consider such a list $(x_A^{n_A}, x_B^{n_B})$. If $n_A = n_B = 1$, there is nothing to prove, so we suppose $n_A + n_B \geq 3$. Applying twice Lemma 2 to the list $(x_A^{n_A+1}, x_B^{n_B+1})$, we obtain $\tilde{f}(x_A^{n_A}, x_B^{n_B}) = f(x_A^{n_A+1}, x_B^{n_B+1})$. Moreover, each value occurs at least twice. A repeated application of Lemma 2 shows then $f(x_A^{n_A+1}, x_B^{n_B+1}) = f(x_A^2, x_B^2)$, from which $f(x_A^{n_A}, x_B^{n_B}) = f(x_A^1, x_B^1)$ follows directly thanks to Lemma 3.

Finally, suppose that the list of occurrences contains only one value x_A, so that $n_A = n$. The trivial case $n_A = n = 1$ is excluded from our model (see Section II). If $n_A > 4$, a repeated application of Lemma 2 shows that $\tilde{f}(x_A^{n_A}) = \tilde{f}(x_A^4)$. If $n_A = 3$, the same result follows from a reverse application of Lemma 2. Since Lemma 3 states that $\tilde{f}(x_A^2) = \tilde{f}(x_A^4)$, we have thus $\tilde{f}(x_A^{n_A}) = \tilde{f}(x_A^2)$ for all $n_A > 1$.

We have thus shown $\tilde{f}(x_A^{n_A}, x_B^{n_B}, x_C^{n_C}, \dots) = \tilde{f}(x_A^1, x_B^1, x_C^1, \dots)$ (or $\tilde{f}(x_A^2)$ in case only one value is present) holds for any computable family of functions, which means they are all order- and multiplicity-independent. ∎

Our proof crucially relies on nodes ignoring their out-degrees, or more formally, on the update rule A_d being independent from the node out-degree. We remark however that this ignorance only plays a role for nodes $1, 2, n-1$, and n in Lemma 2. Moreover, the proof would still hold if the nodes knew that their out-degree was either 1 or 2. So the fundamental limitations of Theorem 1 still hold if nodes know their out-degree up to an uncertainty of 1, or even in the somewhat artificial case where all nodes know their exact out-degree except for four of them that know it up to an uncertainty of 1.

IV. Approximate computation

The fundamental limitations results of the previous section are proved by showing that a computable family of function cannot distinguish an input vector from a "neighboring one" obtained by increasing/decreasing the number of occurrence of a value by 1. This raises the question of the existence of algorithms that approximate certain functions. They might for example converge to $\tilde{f}(x_A^{n_A}, x_B^{n_B}, x_C^{n_C}, \dots) + \epsilon$, where the error ϵ could for example depend on the network but be bounded by a constant independent of n. The following proposition indicates there is very little hope of obtaining approximation algorithms with substantive error bounds valid for all networks: it shows that on some networks, no algorithm can distinguish between certain arbitrarily different inputs.

Proposition 1: For any integer $k > 1$ and distinct values x_A, x_B in an arbitrary nontrivial input set X, there exists a network G on $3 \cdot 2^{k-1} + 4k - 6$ nodes with in-/out-degrees bounded by 2, and input vectors $x, \bar{x}$ such that
(i) x contains $3 \cdot 2^{k-1} - 2$ values x_A and $4k-4$ values x_B, and $\bar{x}$ contains $3 \cdot 2^{k-1} - 2$ values x_B and $4k-4$ values x_A,
(ii) For any algorithm for which the system converges to some y^* on the input x and $\bar{y}^*$ on the input $\bar{x}$, there holds $y^* = \bar{y}^*$.

Proof: Consider the network G shown in Fig. 3, which consists of two parts, L and T, each of them organized in $2k-1$ layers.

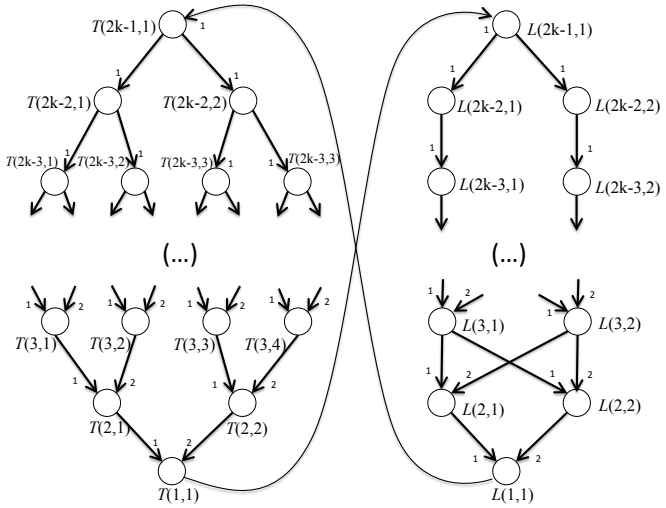

Fig. 3. Network used in Proposition 1. The labels next to each arrow correspond to the port numbers.

In part L, each layer ℓ contains 2 nodes denoted by $L(\ell,1)$ and $L(\ell,2)$, except for layers 1 and $2k-1$ that contain only one node, $L(1,1)$ and $L(2k-1,1)$, respectively. Part L contains thus $4k-4$ nodes. Each node in layers $\ell = 1,\dots k-1$ has two in-neighbors, $L(\ell+1,1), L(\ell+1,2)$, with port numbers 1 and 2 respectively. Nodes in layers $\ell = k,\dots,2k-3$ have one in-neighbor, $L(\ell+1,1)$ for node $L(\ell,1)$ and $L(\ell+1,2)$ for node $L(\ell,2)$, with port number 1. Finally $L(2k-2,1)$ and $L(2k-2,2)$ both have $L(2k-1,1)$ as unique in-neighbor, with port number 1.

In part T, layers $\ell = 1,\dots,k$ have $2^{\ell-1}$ nodes $T(\ell,1),\dots,T(\ell,2^{\ell-1})$. Layers $\ell = k+1,\dots,2k-1$ have $2^{2k-1-\ell}$ nodes $T(\ell,1),\dots,T(\ell,2^{2k-1-\ell})$. One can verify that part T contains $2^k-1+2^{k-1}-1 = 3\cdot 2^{k-1}-2$ nodes. Each node $L(\ell,j)$ of layers $\ell = 1,\dots k-1$ has two in-neighbors, $L(\ell+1,2j-1)$ and $L(\ell+1,2j)$, with port numbers 1 and 2. In layers $\ell = k,\dots,2k-2$, each node $L(\ell,j)$ has one in-neighbor $L(\ell+1,\lceil j/2\rceil)$, with port number 1.

The two parts of the network are interconnected in the following way: $L(1,1)$ is the in-neighbor of $T(2k-1,1)$ and $T(1,1)$ is the in-neighbor of $L(2k-1,1)$, with port number 1 in both cases. Observe that all nodes have in- and out-degrees 1 or 2, and that the total number of nodes in G is $3\cdot 2^{k-1}-2+4k-4 = 3\cdot 2^{k-1}+4k-6$.

We define the inputs x and $\bar{x}$ by letting $x_i = x_A$, $\bar{x}_i = x_B$ for all nodes in part T of the network, and $x_i = x_B$, $\bar{x}_i = x_A$ for all nodes in part B. Due to the number of nodes in L and T, condition (i) is satisfied.

We now consider the execution on G of an arbitrary algorithm starting on the inputs x and $\bar{x}$. By an abuse of notation, we denote by $\bar{b}_i, \bar{y}_i$ and $\bar{z}_i$ the value of b_i, y_i, and z_i in the execution starting on the input $\bar{x}$. We show by induction that for any layer ℓ, $t \geq 0$ and j_1, j_2 for which these expressions are well defined, there holds:

(i) $[\bar{b},\bar{y},\bar{z}]_{L(\ell,j_1)}(t) = [b,y,z]_{T(\ell,j_2)}(t)$,
(ii) $[b,y,z]_{L(\ell,j_1)}(t) = [\bar{b},\bar{y},\bar{z}]_{T(\ell,j_2)}(t)$.

In other words, nodes cannot distinguish a situation where they are in part T (resp. L) and the input was x from a situation where they are in part L (resp. T) and the input was $\bar{x}$.

The two conditions clearly hold at time $t=0$ due to the initialization of the system and to the way that x and $\bar{x}$ are constructed. We now assume they hold at time t and prove that condition (i) then holds at time $t+1$. The proof of condition (ii) follows a parallel argument and is omitted.

We first show that nodes involved in each instantiation of condition (i) receive exactly the same messages. Observe that any node in layers $\ell = 1,\dots,k-1$ of parts L and T has two in-neighbors, both in the next layer $\ell+1$ of the same part, with port numbers 1 and 2. Any node in layers $\ell = k,\dots,2k-2$ has one in-neighbor, in the next layer of the same part. It follows moreover from condition (i) that every in-neighbor of nodes in layer $\ell = 1,\dots,2k-2$ of part L has sent in the execution starting on $\bar{x}$ the same message as did every in-neighbor of nodes in the same layer in part T in the execution starting on x. We now consider the special case of layer $2k-1$. Node $L(2k-1,1)$ has $T(1,1)$ as in-neighbor, and node $T(2k-1,1)$ has $L(1,1)$ as in-neighbor. By condition (ii) at time t, the message broadcast by $T(1,1)$ at time t of the execution starting on $\bar{x}$, and the message broadcast by $L(1,1)$ at time t of the execution starting on x, are the same.

So, we have seen that all nodes in layer ℓ of part L receive the same broadcast message in the execution starting on $\bar{x}$ as do the nodes in layer ℓ of part T in the execution starting on x. It follows moreover from condition (i) that they have the same internal state and estimated answer, so that they compute by (1) exactly the same new values, which establishes condition (i) at time $t+1$. Condition (ii) is proved in the same way.

To conclude the proof, observe that a particular case of condition (i) is $\bar{y}_{L(1,1)}(t) = y_{T(1,1)}(t)$. So if the algorithm converges to y^* when starting on x and $\bar{y}^*$ when starting on $\bar{x}$, we have

$$y^* = \lim_{t\to\infty} y_{T(1,1)}(t) = \lim_{t\to\infty} \bar{y}_{L(1,1)}(t) = \bar{y}^*.$$

Thus, the algorithm produces thus the same output on x and $\bar{x}$. ■

Proposition 1 shows that any approximation scheme would in some cases compute the same values for inputs where the number of occurrences of certain values can be arbitrarily different. Moreover, the proof is based on an example where the two indistinguishable situations involve the same network, in which all out-degrees are either 1 or 2. Its conclusion holds thus *even if the algorithm depends on the network on which it is deployed* (but not on the specific node), and on an estimate of the node out-degree whose error is bounded by 1.

Note that Proposition 1 could be extended to a larger variety of inputs; its proof can indeed directly be applied as long as all nodes in the same layer of L have the same input in x as the nodes in T have in $\bar{x}$, and vice-versa.

V. Applications and examples

A. Minimum computation

The family of function that returns the minimum of the x_i (assuming X admits a total order) is order- and multiplicity-independent, and Theorem 1 does thus not contradict its computability. In fact, it can be easily computed by a classical flooding algorithm: At time 0 every node initializes y_i to its own value x_i and broadcasts it. At each subsequent time, each node sets the new value of y_i to be the minimum of its previous y_i and of the values it received from its in-neighbors. One can verify that $y_i = \min x_i$ holds for every i after at most n time steps. Note that this requires the set of possible broadcast messages B to be equal to the input set X.

B. Voting, Averaging, and Statistics

In a voting problem, every node makes a choice x_i among a finite number (at least two) of options and the goal is to determine the majority vote, i.e., a value taken by the largest number of nodes i, perhaps together with a lexicographic tie-breaking rule, see, e.g., [2]. The corresponding functions clearly depends on the number of occurrences of the values and are thus not computable in our model. Moreover, Proposition 1 shows it is sometimes even impossible to distinguish a situation of almost consensus in favor of one option from a situation of almost consensus in favor of another one.

On the other hand, determining whether there is a full consensus is order- and multiplicity-independent, as it corresponds to checking whether the cardinality of the set of input values is 1, which can be decided, for example, by a flooding algorithm.

Many works in distributed computation address the problem of averaging node values in a distributed manner (see, e.g., [5], [10], [14]). This can be achieved in many different ways when communications are symmetric [17] and/or when nodes know their out-neighbors or at least their cardinality [6]. But the average is not an order- and multiplicity-independent function because it depends on the multiplicity of the different values. It is therefore not computable in our model. The same holds true for almost all statistics, including the variance, the median.

C. Distributed Optimization

Distributed optimization of a sum of functions is another topic that has attracted interest recently; see, e.g., [1], [7], [12], [13], [15], [16]) : Each node has a function $g_i : \Re^m \to \Re$ with suitable convexity properties, and the goal is to find in a distributed manner a minimizer of the sum of these functions:

$$y^* = \arg\min_y \sum_{i=1}^{n} g_i(y).$$

To translate this problem in our context, we take the set of functions $\Re^m \to \Re$ as the input set X, and $\Re^m$ as the output set Y. The input x_i of the nodes are the functions g_i, and the family of functions $(f_n)_{n=1,2,\dots}$ sends the sequences of functions $(g_1, g_2, \dots, g_n)$ to a value in $\Re^m$ that minimizes their sum. This function is not order- and multiplicity-independent, because, in general, $\arg\min(2g_A + g_B) \neq \arg\min(g_A + 2g_B)$, unless the assumptions on the g_i are so strong that they make the problem trivial. Therefore, the distributed optimization of a sum of functions cannot be achieved in our model.

D. Set intersections

Suppose that every node has a set $S_i \subset \Re^m$, and the goal is to find a vector $y \in \Re^m$, in the intersection of these sets. This does not directly correspond to the computation of a function since the intersection of the S_i may contain more than one point, but the problem can be slightly modified to obtain a unique solution, resulting in a well-defined function. The node inputs are here the S_i. The intersection of a collection of sets is order- and multiplicity-independent, $S_i \cap S_i = S_i$, and such functions satisfy therefore the necessary condition of Theorem 1 for being computable in our model. We refer in particular to Mou *et al.* [11] for a broadcast-compatible algorithm that finds a point in the intersection of affine sets of the form $\{u : A_i x = b_i\}$.

VI. Networks and (stochastic) matrices

In many works on distributed algorithms or in control theory, the network is described by a matrix $A \in \Re^{n\times n}$. Node i can then use the value $A_{ij}x_j$ for each of its neighbor j, or sometimes only the weighted average $\sum_j A_{ij}x_j$, or the weighted sum of differences $\sum_{j\neq i} A_{ij}(x_j - x_i)$, with the x_j often being real numbers or vectors. These matrices are often further assumed to have some specific properties, such as their columns and/or rows summing to one [5], [7], [10], [15].

In the context of distributed algorithms, these values A_{ij} should in our opinion be considered as part of the algorithm or as additional information provided by the designer, as opposed to being inherent to the network. Indeed, while the possibility for two nodes to communicate is an objective fact that may depend on physical or technological constraints (or designer choice), no physical or technological constraint forces a specific value A_{ij} or enforces conditions on these values, except possibly in the cases where the interactions consist in exchanging actual matter or energy. In particular, as soon as communication is performed via exchange of messages or via measurements, any node is free to multiply the information it receives by an arbitrary constant.

Computation with such matrices A does not directly fit our model, but can be cast in our model in several ways:

A_{ij} as external information: A first possibility is to assume the A_{ij} are given as additional information to the nodes when the network is established. This is not allowed in our model and is akin to providing nodes with their exact out-degrees. Thus, Theorem 1 does not apply here.

A_{ij} selected by j: Second, one could assume that the emitter j chooses A_{ij} at time 0 and then simply broadcasts the value A_{ij} (or alternatively, multiplies the value of all its

messages by A_{ij}). This is allowed in our model, but since j broadcasts a unique message, all its out-neighbors i should have the same $A_{ij} =: A_j$, which j needs to have computed at time 0 using only its input value and its in-degree. It is therefore impossible to chose the A_{ij} in a way guaranteeing any nontrivial condition involving the columns of A such as $\sum_i A_{ij} = d_j^+ A_j = 1$, as this would require node j to know its out-degree d_j^+ at time 0. It is also impossible to guarantee conditions on the rows of A such as $\sum_j A_{ij} = 1$ and a fortiori, $\sum_i A_{ij} = d \sum_i A_{ji}$, as each node j should then have information about the choices of the other in-neighbors of its out-neighbor i. So algorithms relying on such conditions on rows or columns cannot be cast in our model in that way, and Theorem 1 does not apply to them.

A_{ij} selected by i: In this case, node j broadcasts x_j and the receiver node i selects at time 0 a weight A_{ij}, which is again allowed in our model. In that case, constraints on the columns of A cannot be guaranteed a priori, because they involve choices made by the different out-neighbors of node j, which generally have no way of directly communicating. Algorithms relying on constraints on the columns of A can thus not be cast in our model, and there is thus no contradiction between Theorem 1 and such algorithms successfully computing more complex functions.

On the other hand, constraints on the rows of A can be guaranteed, as all the elements of the i^{th} row of A are chosen by the same node i. In particular a classical consensus algorithm of the form $x_i(t+1) = \sum A_{ij} x_j(t)$ with $A_{ij} \geq 0$ and $\sum_j A_{ij} = 1$ can be implemented in our model. But then, Theorem 1 shows that such an algorithm must compute a order- and multiplicity-independent function if it computes a function at all. Proposition 1 shows moreover that for any way of choosing the A_{ij} that only uses the information available to the nodes, such algorithms are unable to distinguish between situations where, for example, $x_i = 1$ for almost all nodes, and $x_i = 0$ for very few of them, from reverse situations where $x_i = 0$ for almost all nodes and $x_i = 1$ for very few of them.

VII. Sufficient conditions for computability

Theorem 1 shows that being order- and multiplicity-independent is necessary for a family of functions to be computable. One could naturally ask whether it is also sufficient. The answer to this question depends on whether the sets B of possible broadcast messages and Z of internal memory states are given as constraints or can be chosen by the designer.

If Z and B can be freely chosen, a flooding algorithm allows computing any order- and multiplicity-independent function in finite time, see, e.g., [3]. Every node would keep broadcasting its value x_i and the set of values it has already received from its in-neighbors, taking only once into account those appearing multiple times. After at most n steps, it would know all the values present in the network and could directly compute the value of the function. Since nodes do not know n, it would actually update at every time-step an estimate of the answer based on the information already received, and this estimate would reach the correct value in at most n time steps.

But, this approach requires being able to store and broadcast simultaneously all the values appearing in the network (Actually being able to send one element of X at the time is sufficient, as nodes could just sequentially send all the values they have received one by one). Depending on the nature of the set B of possible broadcast messages and Z of internal memory states, this may not always be possible. For example in works on distributed optimization where each node i holds a local function g_i, it is reasonable to assume that only vectors of real numbers can be transmitted and not the whole function. Flooding would thus be impossible in such a context. Another issue is related to the sizes of the sets. Suppose all sets are finite. For very large n, the flooding algorithm may require storing information about the presence or absence in the network of all possible input values in X, which requires a number of bits proportional to the cardinality of X, and thus a cardinality $\mathrm{card}(Z) \geq 2^{\mathrm{card}(X)}$.

To summarize, if the designer can freely chose Z and B, including infinite sets, then every order- and multiplicity-independent family of functions is computable. But if there are cardinality limitations on the sets Z and B, then the exact conditions for computability may be much more complex, and are beyond the scope of this work.

VIII. Conclusions

We have analyzed a model of distributed computation in synchronous deterministic networks where nodes can only broadcast the same message to all their out-neighbors, and do not (initially) know how many out-neighbors they have. We have shown that only a restricted class of functions on the node inputs can be computed on such networks: the order- and multiplicity-independent functions, whose value only depend on which input values are present in the system, but not on how many times they appear. Our arguments can be directly extended to mixed systems involving both wired and broadcast communications.

These results shows that in order to be able to compute more general functions of the node inputs, including most of those mentioned in Section V, the system should violate at least one of our assumptions in Section II. Moreover, *the algorithm should rely on the elements that violate those assumptions*. In particular, the algorithm should explicitly or implicitly use (a) the node out-degree (or some equivalent information), (b) node identifiers, (c) randomization, or (d) asynchronous updates (e.g., rely on some specific properties of the sequence of updates or the random law governing it).

The precise way of using these aspects and how much each of them would help enlarging the class of computable functions are beyond the scope of this work.

References

[1] Mohammad Akbari, Bahman Gharesifard, and Tamás Linder. Distributed online convex optimization on time-varying directed graphs. *preprint:* `http://www.mast.queensu.ca/~bahman/2015-MA-BG-TL.pdf`, 2015.

[2] Florence Bénézit, Patrick Thiran, and Martin Vetterli. The distributed multiple voting problem. *IEEE Journal of Selected Topics in Signal Processing*, 5(4):791–804, 2011.

[3] Paolo Boldi and Sebastiano Vigna. Computing vector functions on anonymous networks. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, page 277, 1997.

[4] Paolo Boldi and Sebastiano Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1):21–66, 2002.

[5] Kai Cai and Hideaki Ishii. Average consensus on arbitrary strongly connected digraphs with time-varying topologies. *IEEE Transactions on Automatic Control*, 59(4):1066–1071, 2014.

[6] Alejandro D Dominguez-Garcia and Christoforos N Hadjicostis. Distributed matrix scaling and application to average consensus in directed graphs. *IEEE Transactions on Automatic Control*, 58(3):667–681, 2013.

[7] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. *IEEE Transactions on Automatic control*, 57(3):592–606, 2012.

[8] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. In *Proceedings of the 2012 ACM Symposium on Principles of distributed computing*, pages 185–194, 2012.

[9] Julien M Hendrickx, Alex Olshevsky, and John N Tsitsiklis. Distributed anonymous discrete function computation. *IEEE Transactions on Automatic Control*, 56(10):2276–2289, 2011.

[10] Franck Iutzeler, Philippe Ciblat, and Walid Hachem. Analysis of sum-weight-like algorithms for averaging in wireless sensor networks. *IEEE Transactions on Signal Processing*, 61(11):2802–2814, 2013.

[11] Shaoshuai Mou, Ji Liu, and A. Stephen Morse. A distributed algorithm for solving a linear algebraic equation. *preprint arXiv:1503.00808v1*, 2015.

[12] Angelia Nedić and Alex Olshevsky. Stochastic gradient-push for strongly convex functions on time-varying directed graphs. *preprint arXiv:1406.2075*, 2014.

[13] Angelia Nedić and Alex Olshevsky. Distributed optimization over time-varying directed graphs. *IEEE Transactions on Automatic Control*, 60(3):601–615, 2015.

[14] Attilio Priolo, Andrea Gasparri, Eduardo Montijano, and Carlos Sagues. A distributed algorithm for average consensus on strongly connected weighted digraphs. *Automatica*, 50(3):946–951, 2014.

[15] Konstantinos I Tsianos, Sean Lawlor, and Michael G Rabbat. Push-sum distributed dual averaging for convex optimization. In *Proceedings of the 51st IEEE Conference on Decision and Control (CDC)*, pages 5453–5458, 2012.

[16] Chenguang Xi, Qiong Wu, and Usman A Khan. Distributed gradient descent over directed graphs. *preprint*, 2015.

[17] Lin Xiao, Stephen Boyd, and Seung-Jean Kim. Distributed average consensus with least-mean-square deviation. *Journal of Parallel and Distributed Computing*, 67(1):33–46, 2007.

[18] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks I. Characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996.