An Effective Characterization of Computability in Anonymous Networks

Paolo Boldi[1] and Sebastiano Vigna[1]

Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Italy

Abstract We provide effective (i.e., recursive) characterizations of the relations that can be computed on networks where all processors use the same algorithm, start from the same state, and know at least a bound on the network size. Three activation models are considered (synchronous, asynchronous, interleaved).

1 Introduction

The question concerning which problems can be solved by a distributed system when all processors use the same algorithm and start from the same state has a long story: it was firstly formulated by Angluin [Ang80], who investigated the problem of establishing a "center". She was the first to realize the connection with the theory of *graph coverings*, which was going to provide, in particular with the work of Yamashita and Kameda [YK96], several characterization for problems which are solvable under certain topological constraints. Further investigation led to the classification of computable functions [YK96,YK98,ASW88,Nor96], and allowed to eliminate several restrictions (such as bidirectionality, distinguished links, synchronicity,...) [DKMP95,BCG+96,BV97a].

Few years ago, while lecturing at the Weizmann Institute about possibility and impossibility results for function computation [BV97a], we were asked by Moni Naor whether our results could be extended to the computation of arbitrary *relations* (as opposed to the indecidability results of [NS95]). Of course, this is the most general case of a distributed task: all classical problems such as election, topology reconstruction etc., can be seen as the computation of a specific relation.

The present paper answers positively to this question. We provide a *proof technique* that allows to show whether an anonymous algorithm that computes a given relation exists under a wide range of models. Moreover, the proof technique is *effective*: if the class of networks under examination is finite, and the relation to be computed is finite, then the technique turns into a recursive procedure that provides either an anonymous algorithm computing the relation, or a proof of impossibility.

Of course, results about specific relations (such as the one definining the election problem or topology reconstruction) are already known, at least for certain models. Here we complete the picture by providing a characterization of the relations that can be computed on a wide range of models when (as usually assumed) a bound on the network size is known. (If such a bound is not known, a completely different approach is needed, as shown in [BV99].)

Our results are mainly of theoretical interest, because of the large amount of information exchanged by the processors. Moreover, complexity issues are not taken into

consideration, as opposed, for instance, to [ASW88,ANIM96]. We rather concentrate on general decidability properties that hold under any assumption of knowledge or of communication primitives (broadcast, point-to-point, etc.).

Our networks are just directed graphs coloured on their arcs (information such as processor identity, communication models etc. can be easily encoded on the arc colouring [BCG$^+$96]), and each processor changes its state depending on its previous state and on the state of its in-neighbours. The problem specification is given by a certain relation between the inputs and the outputs of the processors, and we allow the relation to depend on the network, so that problems like topology reconstruction can be specified.

We consider three known activation models: asynchronous (at each step, we can activate any subset of enabled processors), synchronous (all processors) and interleaved (a.k.a. *central daemon*—exactly one processor), and we give characterization theorems based on the notion of *graph fibration*, a weakening of the notion of covering used in Angluin's paper that also subsumes the concept of similarity of processors introduced in [JS85]. Moreover, the characterizations are stated in a completely uniform way across the activation models—the only change is in the family of fibrations considered.

We remark that the synchronous model is computationally equivalent to asynchronous FIFO networks with finite time delivery (which were the original reason behind the study of the model [YK96]).

The motivation for the study of impossibility results in anonymous networks stems also from questions about self-stabilizing systems: as already noted in [SRR95], and exploited in [BV97b], an impossibility result for anonymous networks gives an impossibility result for a uniform self-stabilizing system (as one of the choices of the adversary is setting all processors to the same state).

We begin by introducing a simple finite class of networks that is used to exemplify our (fairly abstract) characterization theorems. Then, we give our main definitions, recall the standard notion of view and introduce the main mathematical ingredient of our proofs—graph fibrations. Finally, we prove our characterization theorem and give some applications. For other examples of applications, the reader should be able to apply easily our characterization theorems to classical problems such as election, topology reconstruction or function computation, getting back the results of our previous papers.

2 A guiding example

As a guiding example, we formulate a problem that, to our knowledge, has not been previosly addressed in the literature about anonymous networks. We are interested in determining whether there exist an anonymous algorithm solving the *majority problem* on a certain class of networks. All processors are given a boolean value (0 or 1) as input, and eventually must choose an output, the same for all processors, that corresponds to the majority of input values (in case of a tie, either value is correct, provided that all processors agree on that value). The algorithm must work on *every* network of the class under examination (informally speaking: processors just know that they live in one of the networks depicted, but they know neither which one exactly, nor which is their position in the network). We shall use the class depicted in Figure 1 as a case study. For simplicity, in the example we restrict to the synchronous case.

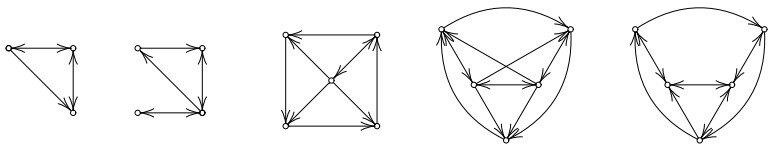

Figure 1. Can you compute majority here?

3 Basic definitions

3.1 Graph-theoretical definitions

A *(directed) (multi)graph* G is given by a nonempty set $N_G = \{1, 2, \ldots, n_G\}$ of nodes and a set A_G of arcs, and by two functions $s_G, t_G : A_G \to N_G$ that specify the source and the target of each arc. An *(arc-)coloured graph* (with set of colours C) is a graph endowed with a colouring function $\gamma : A_G \to C$. We write $i \xrightarrow{a} j$ when the arc a has source i and target j, and $i \to j$ when $i \xrightarrow{a} j$ for some $a \in A_G$. Subscripts will be dropped whenever no confusion is possible.

A (in-directed) *tree* is a graph[1] with a selected node, the root, and such that any other node has exactly one directed path to the root.

3.2 The model

In the following, by a *network* we shall always mean a strongly connected coloured graph[2]. The nodes of such a graph are called *processors*.

Computations of a network are defined by a state space and a transition function specifying how a processor must change its state when it is activated. The new state must depend, of course, on the previous state, and on the states of the in-neighbours; the latter are marked with the colour of the corresponding incoming arc (thus, e.g., if all incoming arcs have different colours a processor can distinguish its in-neighbours). For sake of simplicity, though, we discuss the case with just one colour, so that arcs and nodes are in fact not coloured; the introduction of more colours makes no significant conceptual difference.

We also need a function mapping input values to initial states and final states to output values. Formally, a *protocol* P is given by a set X of local states, a distinguished subset $F \subseteq X$ of final states, an input set Υ, an input function $\mathrm{in} : \Upsilon \to X$, an output set Ω, an output function[3] $\mathrm{out} : F \to \Omega$ and a transition function $\delta : X \times X^{\oplus} \to X$ satisfying the constraint that for every $x \in F$, $\delta(x, -) = x$; here $X^{\oplus}$ is the set of finite multisets over X.

[1] Since we need to manage infinite trees too, we assume that the node set of a tree can be **N**.

[2] Colours on the nodes act as identifiers, whereas colours on the arcs define the communication model. Choosing a suitable colouring we can encode properties such as the presence of unique identifiers, distinguished outgoing/incoming links and so on [BCG+96].

[3] The input and output functions will be silently extended componentwise to vectors.

Intuitively, the new state of a processor depends on its previous state (the first component of the cartesian product) and on the states of its in-neighbours (we must use multisets, as more than one in-neighbour might be in the same state). The additional condition simply states that when a processor enters a final state, it cannot change its own state thereafter, whatever input it receives from its in-neighbours.

A *global state* (for n processors, with respect to the protocol P) is a vector $\boldsymbol{x} \in X^n$. Such a state is *final* if $x_i \in F$ for every i.

Given a network G and a global state $\boldsymbol{x}$, we say that processor i is *enabled in* $\boldsymbol{x}$ iff an application of the transition function would change its state. Of course, no processor is enabled in a final state.

An *activation* for G in state $\boldsymbol{x}$ is a nonempty set of enabled processors; the activation is called

- *synchronous* if it contains every enabled processor;
- *interleaved* if it contains but a single enabled processor.

When we need to emphasize that no constraint is required on the set of activated processors, we use the term *asynchronous*.

A *computation* of the protocol P on the network G (with n processors) with input $\boldsymbol{\upsilon} \in \Upsilon^n$ is given by a (possibly infinite) sequence of global states $\boldsymbol{x}^0, \boldsymbol{x}^1, \ldots, \boldsymbol{x}^T(, \ldots)$ and a sequence of sets of processors $A^0, A^1, \ldots, A^{T-1}(, \ldots)$ (called the *activation sequence*) such that:

1. $\boldsymbol{x}^0 = \mathrm{in}(\boldsymbol{\upsilon})$;
2. for all t, A^t is an activation for G in state $\boldsymbol{x}^t$, and $\boldsymbol{x}^{t+1}$ is obtained applying δ to all processors of A^t (the other processors do not change their state).

A computation is called synchronous, interleaved or asynchronous if every activation is such. It is *terminating* if it is finite and its last state is final; in this case, its output is defined as the vector out $\left(\boldsymbol{x}^T\right)$.

Given a class of networks $\mathscr{C}$, a *($\mathscr{C}$-indexed) relation* R is a function associating to each $G \in \mathscr{C}$ a set $R_G \subseteq \Upsilon^n \times \Omega^n$, where n is the number of processors of G. Relations are used to define problems: for instance, if $\Upsilon = \{*\}$ and $\Omega = \{0, 1\}$, the relation containing all pairs $\langle * * \cdots *, b_1 b_2 \cdots b_n \rangle$, where exactly one of the b_i's is nonzero, defines the well-known *anonymous election problem*. Function computations, topology reconstruction etc. can be easily described in a similar way. Note that classes specify knowledge: the larger the class, the smaller the knowledge (the class of all networks corresponds to no knowledge at all; a singleton to maximum knowledge).

Let $\mathscr{C}$ be a class of networks, R be a relation and P a protocol. We say that P *computes* (in a synchronous, interleaved, asynchronous manner resp.) R on $\mathscr{C}$ if for all $G \in \mathscr{C}$ with n processors it happens that for all $\boldsymbol{\upsilon} \in \mathrm{dom}(R_G)$, every maximal (synchronous, interleaved, asynchronous resp.) computation of P on G with input $\boldsymbol{\upsilon}$ is terminating, and its output $\boldsymbol{\omega}$ is such that $\boldsymbol{\upsilon}\ R_G\ \boldsymbol{\omega}$.

3.3 The view construction

A classical tool in the study of anonymous networks is the concept of *view*, introduced for bidirectional networks in [YK96] and extended to the directed case in [BCG$^+$96] (to

be true, the concept of view is already present in the seminal paper [Ang80], partially disguised under the mathematical notion of *graph covering*). The view of a processor is a tree that gathers all the topological information that the processor can obtain by exchanging information with its neighbours.

More formally, the *view* of a processor i in the network G is an in-tree $\widetilde{G}^i$ built as follows:

- the nodes of $\widetilde{G}^i$ are the (finite) paths of G ending in i, the root of $\widetilde{G}^i$ being the empty path;
- there is an arc from the node π to the node π' if π is obtained by adding an arc a at the beginning of π'.

The tree $\widetilde{G}^i$ is always infinite if G is strongly connected and has at least one arc, and there is a trivial anonymous protocol that allows each processor to compute its own view truncated at any desired depth. At step $k+1$ of the protocol, each processor gathers from its in-neighbours their views truncated at depth k, and combining them it can compute its own view truncated at depth $k+1$. The start state is the one-node tree. An example of view construction is given in Figure 2, where we show the first four steps of view construction for the processors of a simple network.

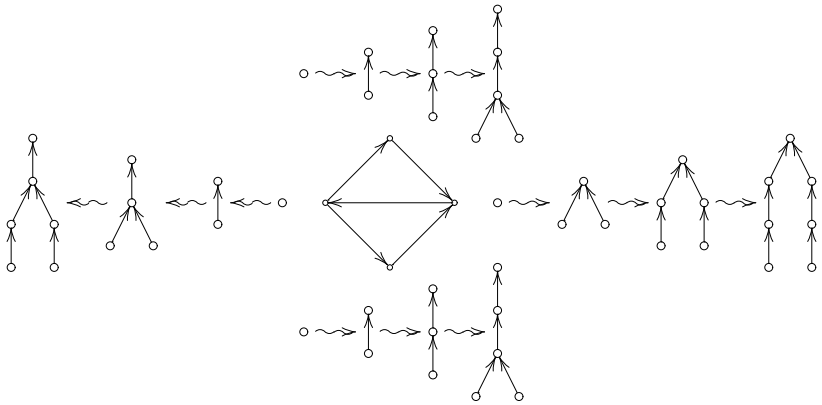

Figure 2. An example of view construction

Clearly, in the view construction process, inputs must be taken into account. That is, the views are really coloured on their nodes by elements on the input set. It is possible that two processor have the same view with a certain choice of inputs, but have a different view with another. For instance, if all processors have different inputs, then all processors get different views, even in the case of a very symmetric topology (e.g., a ring).

The reason why views are important is that the state of a processor after k steps of any anonymous computation may only depend on its view truncated at depth k. Thus, it is of crucial importance to determine which processors of a network possess the same (infinite) view.

3.4 Graph fibrations and the Lifting lemma

The problems stated at the end of the last section can be solved very elegantly using an elementary graph-theoretical concept, that of *graph fibration* [BV]. A fibration formalizes the idea that processors that are connected to processors behaving in the same way will behave alike; it generalizes both the usage of graph coverings in Angluin's original paper [Ang80] and the concept of similarity of processors introduced in [JS85].

Recall that a *graph morphism* $f : G \to H$ is given by a pair of functions $f_N : N_G \to N_H$ and $f_A : A_G \to A_H$ that commute with the source and target functions, that is, $s_H \circ f_A = f_N \circ s_G$ and $t_H \circ f_A = f_N \circ t_G$. (The subscripts will usually be dropped.) In other words, a morphism maps nodes to nodes and arcs to arcs in such a way to preserve the incidence relation. If colours are present (on the arcs or on the nodes) they must be preserved.

Definition 1 A *fibration* between (coloured) graphs G and B is a morphism $\varphi : G \to B$ such that for each arc $a \in A_B$ and for each node $i \in N_G$ satisfying $\varphi(i) = t(a)$ there is a unique arc $\widetilde{a}^i \in A_G$ (called the *lifting of a at i*) such that $\varphi(\widetilde{a}^i) = a$ and $t(\widetilde{a}^i) = i$.

If $\varphi : G \to B$ is a fibration, B is called the *base* of the fibration. We shall also say that G is *fibred (over B)*. The *fibre* over a node $i \in N_B$ is the set of nodes of G that are mapped to i, and will be denoted by $\varphi^{-1}(i)$.

There is a very intuitive characterization of fibrations based on the concept of local isomorphism. A fibration $\varphi : G \to B$ induces an equivalence relation between the nodes of G, whose classes are precisely the fibres of φ. When two nodes i and j are equivalent (i.e., they are in the same fibre), there is a bijective correspondence between arcs coming into i and arcs coming into j such that the sources of any two related arcs are equivalent.

In Figure 3 we sketched a fibration between two graphs. Note that, because of the lifting property described in Definition 1, all black nodes have exactly two incoming arcs, one (the dotted arc) going out of a white node, and one (the continuous arc) going out of a grey node. In other words, the in-neighbour structure of all black nodes *is the same*.

The main *raison d'être* of fibrations is that they allow to relate the behaviour of the same protocol on two networks. To make this claim precise, we need some notation: if $\varphi : G \to B$ is a fibration, and $\boldsymbol{x}$ is a global state of B, we can obtain a global state $\boldsymbol{x}^\varphi$ of G by "lifting" the global state of B along each fibre, that is, $(\boldsymbol{x}^\varphi)_i = x_{\varphi(i)}$ (see Figure 4). Essentially, starting from a global state of B we obtain a global state of G by copying the state of a processor of B fibrewise.

Lemma 1 (Lifting Lemma [BV97a]). *Let $\varphi : G \to B$ be a fibration. Then, for every protocol P and every synchronous computation $\boldsymbol{x}^0, \boldsymbol{x}^1, \ldots$ of P on B, $(\boldsymbol{x}^0)^\varphi, (\boldsymbol{x}^1)^\varphi, \ldots$ is a synchronous computation of P on G.*

The above lemma suggests that if a network can be "collapsed" by a fibration onto another one, then there are certain constraints on its behaviours. More precisely, we can compute the sequence of lifted global states of G by computing the sequence of global states of B, and then lifting the result. In other words, the computation of B "summerizes" the computation of G.

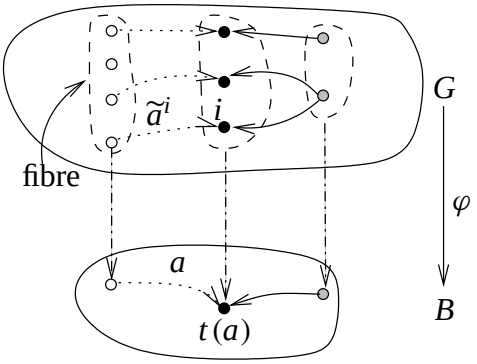

Figure 3. A fibration.

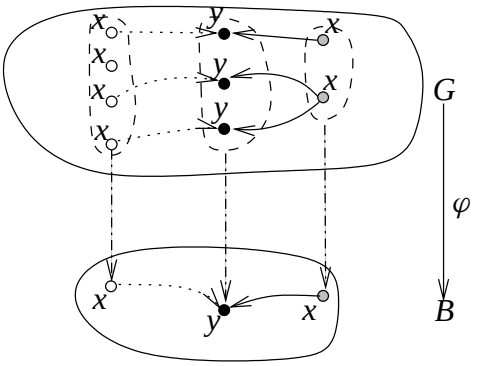

Figure 4. State lifting.

In particular, the computation above terminates in T steps on B iff the lifted computation does the same on G. Moreover, the outputs of the two computations are clearly related: the output of a processor of G is exactly the output of the processor of B it is mapped to by the fibration.

Note that the above lemma does not provide constraints on *all* computations of G. Rather, it works when inputs are assigned fibrewise (or, from a symmetrical point of view, when they are lifted from B). In this case, it tells us that we can really do the computation on B and lift the result at the end.

These considerations may seem trivial when applied to a single network, but things become trickier when a whole class is involved. Indeed, if *two* different networks G and H of the class are fibred over a common base B, then the output of the two networks with respect to inputs that are liftings of the same input of B are inextricably related.

Indeed, these observations are already sufficient to show that *majority is not computable in the class of Figure 1*. In Figure 5 we show that two networks G and H of our class are fibred over a common base B. Suppose by contradiction that you have a protocol computing the majority. If we run this protocol on B with, say, input 0 to a and 1 to b, it will terminate with output ω (0 or 1 are both correct; note that it must terminate, because it terminates on G and H). However, if we lift this input to G and H, ω will not be a correct output for one of them—contradiction.

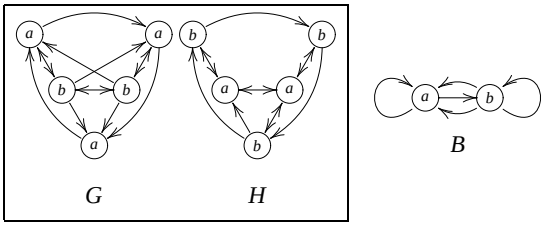

Figure 5. Two networks with a common base.

4 Characterizing solvability for arbitrary classes

Using fibration we have proved an impossibility result for our example. However, there is a missing link: processor can build views, but are constrained in their behaviour by the bases of the network they live in. The connection between these two concepts lies in the notion of *minimum base*.

4.1 Minimum bases

We say that a graph G is *fibration prime* if every fibration from G is an isomorphism, that is, G cannot be collapsed onto a smaller network by a fibration.

Theorem 1 ([BV]) The following properties hold:

1. For every graph G there is (up to isomorphism) exactly one fibration-prime graph $\hat{G}$ such that G is fibred onto (i.e., surjectively) $\hat{G}$.
2. If G and H have a common base B, then $\hat{G} \cong \hat{H}$.
3. Let $\varphi : G \to \hat{G}$; then $\widetilde{G}^i \cong \widetilde{G}^j$ iff $\varphi(i) = \varphi(j)$ (as a consequence, distinct nodes of a fibration prime graph have different views).
4. A fibration-prime graph is uniquely characterized by the set of views of its nodes.

There are many ways to build $\hat{G}$. One is a *partition set* algorithm [BV], similar to the one used for finite-state automaton minimization. On the other hand, one can also take as nodes of $\hat{G}$ the distinct views of G, and put an arc between two views if one view is a first-level child of the second one.

The fibrations from G to $\hat{G}$ are called *minimal*. There is usually more than one minimal fibration, but they must all coincide on the nodes by property (3) of Theorem 1, so we denote with μ_G one of them when the map on the arcs is not relevant.

In Figure 6 we show a number of networks and the corresponding minimum bases. Again, the node component of a minimal fibration is represented by suitably labelling the nodes. Another example of minimum base was given in Figure 5 the graph B was the minimum base of both G and H, that is, $\hat{G} \cong \hat{H} \cong B$.

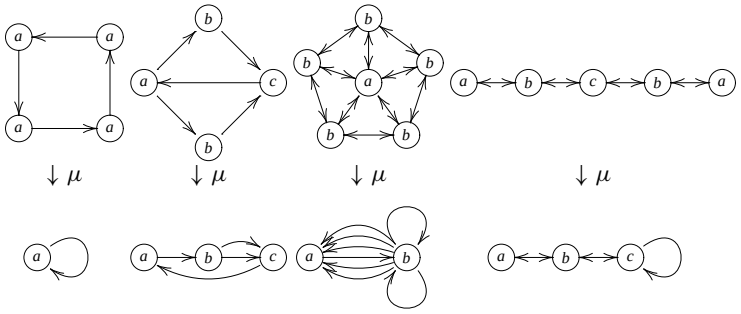

Figure 6. Some examples of minimal fibrations and minimum bases.

The previous theorem highlights the deep link between fibrations and views: two processors have the same view if and only if they lie in the same fibre of μ. Since we know by the Lifting Lemma that processors in the same fibre of a fibration cannot behave differently, this means that on the one hand, processors with the same view will always be in the same state, and on the other hand, if we can deduce $\hat{G}$ from a view we can use the Lifting Lemma to characterize the possible behaviours.

The fundamental fact we shall use intensively in all proofs is that the above considerations, which involve infinite objects (isomorphism of infinite trees, and so on), can be described by means of finite entities using the following theorem:

Theorem 2 ([BV]) Let G be a strongly connected graph and B a fibration-prime graph with minimum number of nodes such that, for some node i of G and node j of B, $\widetilde{G}^i$

and $\widetilde{B}^j$ are isomorphic up to height $n_G +$ (diameter of G): then $B \cong \hat{G}$, and i is mapped to j by all minimal fibrations.

How can a processor use the (apparently unfathomable) previous theorem to build the minimum base? First of all, in $2N - 1$ rounds all processors build their views up to height $2N - 1$, where N is a known bound on the number of processors (and thus $N - 1$ bounds the diameter). Then, they perform locally an exhaustive search for the only graph B satisfying the theorem.[4]

Note that since our processors have additional information given by their inputs, the constructions described here must be performed *taking the inputs into account*. More precisely, views and minimum bases must be constructed "as if" the processors were (additionally) coloured with their inputs. For instance, in Figure 7 we show the minimum base for a 4-cycle with respect to different inputs.

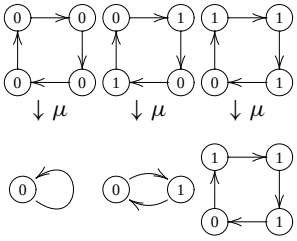

Figure 7. Minimal fibrations of the same graph with respect to different inputs.

4.2 The (a)synchronous case

We are finally ready to characterize the classes of networks on which (a)synchronous computation of a relation R is possible. First of all we notice that, from a computability viewpoint, there is no difference between the synchronous and the asynchronous case, as shown in [BV99]; thus, we restrict our proofs to the synchronous case.

Theorem 3 Let $\mathscr{C}$ be a class of networks of bounded size[5]. Then R is (a)synchronously computable on $\mathscr{C}$ iff for all graphs B and all $\boldsymbol{v} \in \Upsilon^{n_B}$ there is an $\boldsymbol{\omega} \in \Omega^{n_B}$ such that for all $G \in \mathscr{C}$ and all fibrations $\varphi : G \to B$, if $(\boldsymbol{v})^\varphi \in \mathrm{dom}(R_G)$ then $(\boldsymbol{v})^\varphi \; R_G \; (\boldsymbol{\omega})^\varphi$.

Proof. If the conditions of the statement are not satisfied, then there is a graph B and a $\boldsymbol{v} \in \Upsilon^{n_B}$ such that for all $\boldsymbol{\omega} \in \Omega^{n_B}$ there is a $G \in \mathscr{C}$ and a fibration $\varphi : G \to B$ such that $(\boldsymbol{v})^\varphi \in \mathrm{dom}(R_G)$ but not $(\boldsymbol{v})^\varphi \; R_G \; (\boldsymbol{\omega})^\varphi$. By the Lifting Lemma this implies that,

[4] There are of course "smarter" ways to find $\hat{G}$ for specific types of networks; nonetheless, for the general case Theorem 2 gives an optimal bound [BV].

[5] By this we mean that there is an integer N such that all networks in $\mathscr{C}$ have at most N nodes.

whichever protocol we use, at least one computation on some graph G in $\mathscr{C}$ starting from $\mathrm{in}((\boldsymbol{v})^{\varphi})$ will give an output that is incorrect.

Otherwise, each processor determines its minimum base taking into account the inputs, and, using the conditions above, chooses an output value that satisfies R_G (i.e., it looks for the correct output on the minimum base and chooses its own output accordingly).

The characterization implied by statement of the previous theorem may seem to be daunting, but it really boils down to the following simple procedure:

1. Compute the minimum bases of the networks in $\mathscr{C}$ with respect to every possible input assignment.
2. For each base, choose an output $\boldsymbol{\omega}$ satisfying the conditions of the theorem.
3. If the previous step is not possible, then the relation cannot be computed on $\mathscr{C}$; otherwise, the chosen outputs can be put into a table and used by the processors, after the minimum base construction, to choose their output.

All the above steps are finite if $\mathscr{C}$ and R are finite. Thus, the characterization is recursive.

We can now apply the characterization to our original problem. First of all, we have to throw away one of the last two networks: as we already argued (and as easily shown using the above theorem) there is no majority algorithm for the class if both networks of Figure 5 are present. Hence, we restrict to the first four networks of Figure 1

We note that the first three networks in Figure 1 have a different minimum base than the others, as shown in Figure 8: thus, we can consider them separately. Now, a trivial

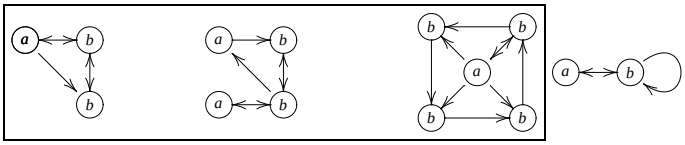

Figure 8. The remaining networks and their minimum base.

application of the procedure we outlined shows that the conditions of Theorem 3 are satisfied: the only subtle point is that if we assign different inputs to the two nodes of the minimum base, there is always a choice for the output that works when lifted to the three networks.

A similar investigation shows, for instance, that election is not solvable in the restricted class of Figure 8: no matter which processor we decide to elect on the base, there will be two processors elected in the second network. If the latter is eliminated, however, election becomes possible.

4.3 The interleaved case

We sketch the ideas behind the characterization of the interleaved case. We say that a fibration is *acyclic* if the subgraphs induced by each fibre are acyclic (except for loops).

Acyclicity guarantees that there is always an interleaved activation working fibrewise, so that the Lifting Lemma can be extended to this case (as done in [BV97a]). More precisely, for each interleaved activation on the base B of a fibration $\varphi : G \to B$ one can find an interleaved activation of G such that processors are activated fibre by fibre. The activation in each fibre starts from a sink in the fibre and continues inductively so that no processor in the fibre is activated before one of its successors in the fibre.

On the positive side, processors use a simple 0–1 game (as shown in [BCG$^+$96] for election) to break asymmetry until they obtain a minimal *acyclic* fibration (that is, a fibration that collapses a network as much as possibly without violating acyclicity). The theory in this case is a bit different, as there are many possible, nonisomorphic, minimal bases. Nonetheless, the statement of the characterization remains the same:

Theorem 4 Let $\mathscr{C}$ be a class of networks of bounded size. Then R is computable with interleaved activation on $\mathscr{C}$ iff for all graphs B and all $\boldsymbol{v} \in \Upsilon^{n_B}$ there is an $\boldsymbol{\omega} \in \Omega^{n_B}$ such that for all $G \in \mathscr{C}$ and all acyclic fibrations $\varphi : G \to B$, if $(\boldsymbol{v})^\varphi \in \mathrm{dom}(R_G)$ then $(\boldsymbol{v})^\varphi \; R_G \; (\boldsymbol{\omega})^\varphi$.

Proof. If the conditions of the statement are not satisfied, then there is a graph B and a $\boldsymbol{v} \in \Upsilon^{n_B}$ such that for all $\boldsymbol{\omega} \in \Omega^{n_B}$ there is a $G \in \mathscr{C}$ and an acyclic fibration $\varphi : G \to B$ such that $(\boldsymbol{v})^\varphi \in \mathrm{dom}(R_G)$ but not $(\boldsymbol{v})^\varphi \; R_G \; (\boldsymbol{\omega})^\varphi$. By the Lifting Lemma (extended to ayclic fibrations and interleaved activations) this implies that, whichever protocol we use, at least one computation on some graph G starting from $\mathrm{in}((\boldsymbol{v})^\varphi)$ will give an output that is incorrect.

With respect to the proof of Theorem 3, we have to show that we can break symmetry in such a way to obtain a fibration $\varphi : G \to B$ which is acyclic. To this purpose, the processors first compute a standard minimal fibration (taking inputs into account); then, processors in the same fibre label themselves either 0 or 1 according to the following rule:

- if the processor is activated and all in-neighbours in its fibre are unlabelled, it labels itself 0;
- if the processor is activated and any in-neighbours in its fibre is labelled, it labels itself 1.

At the end of this "0–1 game", the processors compute again the fibres of a minimal fibration, this time taking into account the 0–1 identifier they possess. This process can be repeated, and can only terminate when the minimal fibration is acyclic; this happens because at each step every nontrivial fibre containing an oriented cycle breaks at least into two pieces (in such a fibre, the first activated processor labels itself 0, and among the processors belonging to an oriented cycle the least activated processor necessarily labels itself 1). Now all processors can select an output using the conditions in the statement.

If we get back to our guiding example, we can note that all networks in Figure 1 are minimal with respect to acyclic fibrations, except for the third, which is acyclically fibred over the first one. Again, a simple case-by-case examination shows that now both majority and election are possible, due to the great desymmetrizing power of interleaved activation.

5 Some easy applications

We give a few theorems outlining more general applications of our results. To our knowledge, these are the first results for these problems (the results apply also to the interleaved model).

Theorem 5 Let $\mathscr{C}$ be the class of all bidirectional networks of size n (that is, every arc has an associated arc in the opposite direction). Then majority is computable on $\mathscr{C}$.

Proof. (Sketch) The linear constraints imposed by bidirectionality determine uniquely the cardinality of the fibres of a fibration once its base is known [BV]. Thus, the output vector $\boldsymbol{\omega}$ in the statement of Theorem 3 can be chosen by counting the actual multiplicity of each input.

We can give a more elementary point of view on the previous theorem: by solving a linear system, each processor can determine the cardinality of all fibres of the minimal fibration. Then, it can compute locally the multiplicity of each input, and thus derive the correct output.

Note that the above theorem is independent of the colouring on the arcs, and thus of the communication model. For instance, it works in broadcast networks without distinguished incoming links.

Theorem 6 Let $\mathscr{C}$ be a class of networks with distinguished outgoing links (that is, every arc has a colour and arcs going out of the same processor get different colours) and bounded size. Then majority is computable on $\mathscr{C}$.

Proof. Since the outgoing links have different colours, all fibrations starting from a network in $\mathscr{C}$ have the property that all fibres have the same cardinality (this happens because they are really *coverings*—see [BV]). Thus, lifting multiplies each input the same number of times; hence, the output vector $\boldsymbol{\omega}$ can be chosen by taking a majority vote on $\boldsymbol{\upsilon}$.

The theorem above would not hold if we required distinguished *incoming* links. A counterexample can be built by colouring all arcs of the networks B of Figure 5 with different colours, and lifting the colours on the networks G and H.

It is interesting to contrast the behaviour of the *counting* problem in the above two cases: each processor must compute how many processor with its own input are present.

Theorem 7 Let $\mathscr{C}$ be the class of all bidirectional networks of size n (that is, every arc has an associated arc in the opposite direction). Then counting is computable on $\mathscr{C}$.

Theorem 8 Let $\mathscr{C}$ be a class of networks with distinguished outgoing links (that is, every arc has a colour and arcs going out of the same processor get different colours) and size n. Then counting is computable on $\mathscr{C}$.

The theorem above does *not* hold if just a bound on the size is known, and, of course, if we required distinguished incoming links. The proof of the last statements should be now an easy exercise for the reader.

5.1 Conclusions

We have only surfaced the range of applications of the general theory of anonymous computation sketched in this paper. For instance, the theory also allows one one to establish the possibility of *weak* computability. A relation is *weakly* computable on a class $\mathscr{C}$ if there is a protocol that works correctly on all networks of the class on which the relation is computable, but stops all processors in a special "impossibility state" in the remaining networks. In other words, the processors either compute the result, or establish in a distributed and anonymous way that this is impossible on specific network they live in (previous literature often confused the strong and weak version of a problem). Of course, in general the classes on which a relation is weakly computable are much larger than in the standard case. It is not difficult to derive from Theorem 3 a necessary and sufficient condition for weak computability.

As we noticed elsewhere, the bound given on the number of steps required to compute a relation (the number of nodes plus the diameter) is optimal, as there are classes of graphs in which topology reconstruction cannot be performed in less than that number of steps.

References

[Ang80] Dana Angluin. Local and global properties in networks of processors. In *Proc. 12th Symposium on the Theory of Computing*, pages 82–93, 1980.

[ANIM96] Nechama Allenberg-Navony, Alon Itai, and Shlomo Moran. Average and randomized complexity of distributed problems. *SIAM Journal on Computing*, 25(6):1254–1267, 1996.

[ASW88] Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. *J. Assoc. Comput. Mach.*, 35(4):845–875, 1988.

[BCG+96] Paolo Boldi, Bruno Codenotti, Peter Gemmell, Shella Shammah, Janos Simon, and Sebastiano Vigna. Symmetry breaking in anonymous networks: Characterizations. In *Proc. 4th Israeli Symposium on Theory of Computing and Systems*, pages 16–26. IEEE Press, 1996.

[BV] Paolo Boldi and Sebastiano Vigna. Fibrations of graphs. *Discrete Math.* To appear.

[BV97a] Paolo Boldi and Sebastiano Vigna. Computing vector functions on anonymous networks. In Danny Krizanc and Peter Widmayer, editors, *SIROCCO '97. Proc. 4th International Colloquium on Structural Information and Communication Complexity*, volume 1 of *Proceedings in Informatics*, pages 201–214. Carleton Scientific, 1997. An extended abstract appeared also as a Brief Announcement in *Proc. PODC '97*, ACM Press.

[BV97b] Paolo Boldi and Sebastiano Vigna. Self-stabilizing universal algorithms. In Sukumar Ghosh and Ted Herman, editors, *Self–Stabilizing Systems (Proc. of the 3rd Workshop on Self–Stabilizing Systems, Santa Barbara, California, 1997)*, volume 7 of *International Informatics Series*, pages 141–156. Carleton University Press, 1997.

[BV99] Paolo Boldi and Sebastiano Vigna. Computing anonymously with arbitrary knowledge. In *Proc. 18th ACM Symposium on Principles of Distributed Computing*, pages 181–188. ACM Press, 1999.

[DKMP95] Krzysztof Diks, Evangelos Kranakis, Adam Malinowski, and Andrzej Pelc. Anonymous wireless rings. *Theoret. Comput. Sci.*, 145:95–109, 1995.

[JS85] Ralph E. Johnson and Fred B. Schneider. Symmetry and similarity in distributed systems. In *Proc. 4th conference on Principles of Distributed Computing*, pages 13–22, 1985.

[Nor96] Nancy Norris. Computing functions on partially wireless networks. In Lefteris M. Kirousis and Evangelos Kranakis, editors, *Structure, Information and Communication Complexity. Proc. 2nd Colloquium SIROCCO '95*, volume 2 of *International Informatics Series*, pages 53–64. Carleton University Press, 1996.

[NS95] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.

[SRR95] Sandeep K. Shukla, Daniel J. Rosenkrantz, and S.S. Ravi. Observations on self-stabilizing graph algorithms for anonymous networks. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 7.1–7.15, Las Vegas, 1995. University of Nevada.

[YK96] Masafumi Yamashita and Tiko Kameda. Computing on anonymous networks: Part I—characterizing the solvable cases. *IEEE Trans. Parallel and Distributed Systems*, 7(1):69–89, 1996.

[YK98] Masafumi Yamashita and Tiko Kameda. Erratum to computing functions on asynchronous anonymous networks. *Theory of Computing Systems (formerly Math. Systems Theory)*, 31(1), 1998.