Universal Dynamic Synchronous Self–Stabilization*

Paolo Boldi Sebastiano Vigna
Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Italy

Abstract

We prove the existence of a "universal" synchronous self-stabilizing protocol, that is, a protocol that allows a distributed system to stabilize to a desired nonreactive behaviour (as long as a protocol stabilizing to that behaviour exists). Previous proposals required drastic increases in asymmetry and knowledge to work, whereas our protocol does not use any additional knowledge, and does not require more symmetry-breaking conditions than available; thus, it is also stabilizing with respect to dynamic changes in the topology. We prove an optimal quiescence time $n + D$ for a synchronous network of n processors and diameter D; the protocol can be made finite state with a negligible loss in quiescence time. Moreover, an optimal $D + 1$ protocol is given for the case of unique identifiers. As a consequence, we provide an effective proof technique that allows to show whether self-stabilization to a certain behaviour is possible under a wide range of models.
Keywords: self-stabilization, anonymous networks, graph fibrations, synchronous systems.

1 Introduction

A system is self-stabilizing if, for every initial state, after a finite number of steps it cannot deviate from a specified behaviour. Self-stabilization of distributed systems was introduced by Dijkstra in his celebrated paper [13], and has become since an important framework for the study of fault-tolerant computations.

The adversarial choice of the initial state makes it extremely difficult to devise self-stabilizing protocols. To overcome this problem, there have been some recent attempts in the literature to build "general" self-stabilizing protocols (also called *extensions*), namely, superimposed distributed protocols that make the underlying behaviour self-stabilizing, or to solve classical problems (election, spanning tree construction, etc.) in a self-stabilizing way. For instance, Katz and Perry [19] propose a combination of self-stabilizing global snapshots and resets to this purpose; Afek, Kutten and Yung [4] and Awerbuch, Patt–Shamir and Varghese [6] use local conditions to initiate a global or a local correction; Awerbuch and Varghese [7] use a *resychronizer* to restart a non-self-stabilizing protocol until stabilization; Afek and Dolev [2] collect *pyramids of views* of the system to detect incoherences and correct the system behaviour; in a different vein, Dolev, Israeli and Moran [15, 17] use randomization to assign unique identifiers.

However, all previously mentioned methods require additional asymmetry and knowledge: the self-stabilizing extension of [19] requires a distinct processor and knowledge of the number of processors in the network, which may not be available. The protocol described in [4] needs unique identifiers or the power of randomization, whereas [6] requires the ability to distinguish incident links. The resynchronizer [7] needs point-to-point communication and a bound on the diameter to work. The tecniques described in [2] require the knowledge of the topology of the network. In all the above cases, bidirectionality of the links is an essential feature.

It is a major open problem (if only of theoretical importance) to establish (or prove impossible) the existence of a *universal* deterministic self-stabilizing protocol: by "universal" we mean that such a protocol should be able to self-stabilize to any behaviour for which a self-stabilizing protocol exists, under given conditions of asymmetry and knowledge. In particular, the protocol should be *uniform*, that is, the same for all processors, unless the presence of (possibly unique) identifiers is explicitly assumed.

In this paper we prove a surprising and apparently unnatural result, viz., that such a protocol exists for synchronous dynamic[1] networks, and that it has tight quiescence time[2] $n + D$, where n is the number of processors in the system and D its diameter. It is possible to modify the protocol so that it becomes finite state, with a negligible loss in quiescence time.

A widely accepted *tenet* about self-stabilization is that it is a difficult *coordination* problem, independently of the resources available. We show that, on the contrary, unless strong bounds are imposed on resources and quiescence time, the coordination part of the problem *is trivial* (at least in the synchronous case), in the sense that it can be completely solved once for all (there is a universal protocol for it).

The model we use is very general, as a network is just a directed graph (possibly with parallel arcs and loops) coloured on the nodes and on the arcs. The former colouring can be used to encode identity information, whereas the latter represents the degree of *port awareness* of the network (i.e., the ability of each processor to distinguish among its incident links). Our results do not make *any* assumption on the particular class of networks under study: in particular, we assume neither bidirectionality (and this is a major difference with existing literature, as a processor cannot communicate immediately with its in-neighbours), nor a particular distribution of identifiers (such as unique identifiers, or a unique distinguished processor).

On the other hand, we use a synchronous activation model, in which all processors take a step at the same time. Moreover, at each step the new state of a processor can depend arbitrarily on the states of its in-neighbours, so there is no bound on the amount

*Part of the results of this paper appeared in the Proceedings of the 3rd Workshop on Self–Stabilizing Systems, Santa Barbara, California, 1997 [11].

[1] With *dynamic* we mean that our protocols tolerate modifications to the network topology, such as insertion or deletion of processors and links.

[2] The *quiescence time* of a self-stabilizing protocol is the maximum number of steps required to restart a correct behaviour after a transient fault.

of information exchanged along the links.

Our protocols tolerate dynamic changes in the network structure: at any time an adversary can change the topology of the network, or corrupt the identifiers of the processors, as long as the resulting network fits the knowledge assumed (the last characteristic, in particular, was not—as far as we know—achieved by any protocol). After the quiescence time, the desired behaviour will restart.

To prove our results, we exploit a mix of classical techniques for self-stabilization, and a series of results from the theory of anonymous networks. A network is *anonymous*[3] if all processors are identical and start from the same initial state; these assumptions make them indistinguishable. The systematic study of such networks was initiated by Angluin [5], and continued by Yamashita and Kameda; there is now a rather complete characterization of what is computable on such networks for several problems, such as election [22, 8] and function computation [21, 23, 10]. In fact, these characterizations work even if the processors have identifiers, as long as the initial states are the same, and this observation is the key to our proofs.

There is clearly a link between anonymous networks and self-stabilizing systems, since one of the possible choices for an adversary is to start all processors from the same state. Thus, in a sense that we will make precise, there is no way a network can self-stabilize to something that it cannot compute anonymously. We obtain our results by turning the classical algorithms from the theory of anonymous networks into self-stabilizing protocols.

The (nonreactive) behaviours we consider are modeled by a suffix-closed set of infinite sequences of (excerpts of) global states; the set depends on the network, so problems like topology reconstruction can be specified. The definitions of self-stabilization we have adopted here[4] requires that in finite time the behaviour of the network belongs to the selected set.

Our results are mainly of theoretical interest, because of the large amount of information exchanged by the processors. Nonetheless, they provide for the first time general upper and lower bounds for self-stabilization. Moreover, we obtain a characterization of the behaviours to which self-stabilization is possible. As a result, we provide a *proof technique* that allows to show whether self-stabilization to a certain behaviour is possible under a wide range of models. Moreover, the proof technique is *effective*: if the class of networks under examination is finite, and the desired behaviour is finite state, then the technique turns into a recursive procedure that provides either a self-stabilizing protocol for the behaviour, or a counterexample.

In Section 2 we discuss informally a simple possibility problem whose solution will guide the development of the paper. In Section 3 we define the basics of the model of computation. Then, in Section 4 we discuss briefly some known notions from the theory of anonymous networks, in particular that of *view*, and we show how to make the view construction self-stabilizing. On the other side, in Section 5 we provide corresponding negative results by stating in graph-theoretic terms some necessary conditions for self-stabilization; this conditions turn out to match our positive results about self-stabilizing view construction, giving the first characterization of attainable behaviours and solving the guiding problem. Section 6 delves into the details of the proof that the self-stabilizing view construction protocol can be made finite state without losing in generality, thus providing another characterization, this time for finite-state attainable behaviours. Finally, in Section 7 we approach the very special, yet extremely relevant, case of networks with unique identifiers, showing a fundamental gap with the general case, as the bounds for this case (which are still tight) depend only on the diameter and not on the number of processors. We conclude the paper with a section of examples, which show how our theory allows one to characterize exactly the knowledge necessary to perform a certain task; moreover, we describe informally the protocols obtained specializing the universal ones.

2 A guiding example

Suppose one is interested in determining whether there exist a self-stabilizing leader election protocol that works in the class of networks depicted in Figure 1. In other words, you should produce a synchronous *uniform* protocol (i.e., the program must be the same for all processors) that self-stabilizes to an election global state; the protocol must work on *every* network of the class (informally speaking: the 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). Questions of

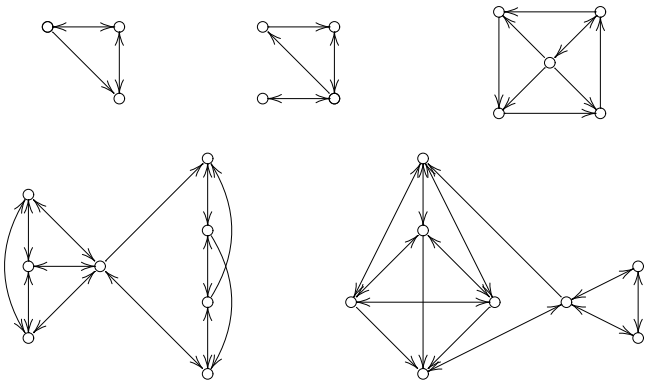


Figure 1: Can you do election here?

this kind have been often considered in the literature for specific problems, specific models and specific network classes (a classic result says, for instance, that under a serialized activation election is possible in rings of prime size). As stated in the introduction, the main goal of this paper is to "factor out" the common part of these results, and give a completely general criterion to establish such (im)possibility results. We shall use the example depicted in Figure 1 as a case study.

[3] The terms *uniform* and *anonymous* are often used as synonyms in the literature. However, we propose to use the former when all processors are identically programmed (said otherwise, they have no identifiers), and the latter when *in addition* all processors are forced to start from the same state; we think that it is important to tell *potential* from *actual* indistinguishability.

[4] There seems to be a certain disagreement in the literature about the exact definition of self-stabilization. For instance, [19], [12] and [4] use different and increasingly more restrictive definitions.

3 Basic definitions and preliminaries

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. A *(arc- and node-)coloured graph* (with set of colours C) is a graph endowed with a colouring function $\gamma : N_G + A_G \to C$ (we denote with $+$ disjoint union). 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 $a \in A_G$; we often use the shorthand $i \nrightarrow j$ for "$i \to j$ or $i = j$". We denote with $d_G(i, j)$ the *distance* between two nodes i and j of a graph G, and with D_G its diameter. Subscripts will be dropped whenever no confusion is possible.

A (in-directed) *tree* is a graph[5] with a selected node, the root, and such that every node has exactly one directed path to the root. If T is a tree, we write $h(T)$ for its *height* (the length of the longest directed path). In every tree we consider in this paper, all maximal paths have length equal to the height. We write $T \restriction k$ for the tree T truncated at height k, that is, we eliminate all nodes at distance greater than k from the root.

Trees are partially ordered[6] by *prefix*, that is, $T \leq U$ iff $T \cong U \restriction h(T)$, where $\cong$ denotes isomorphism; this partial order is augmented with a bottom element $\perp$, with $h(\perp) = -1$ by definition (so h is strictly monotonic). The infimum in this partial order, denoted by $\wedge$, is the tallest common prefix (or $\perp$ if no common prefix exists). The supremum between T and U exists iff T and U are comparable.

We remark a nontrivial property relating order and height: if $T = U \restriction k$ then

$$h(T \wedge V) = \min\{h(U \wedge V), h(T)\} = \min\{h(U \wedge V), k\}. \quad (1)$$

3.2 The model

In the following, by a *network* we shall always mean a strongly connected coloured graph. 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 colours of the corresponding processor and incoming arc (thus, e.g., if all incoming arcs have different colours a processor can distinguish its in-neighbours, even if they do not possess distinct colours). Moreover, the new state must depend also on the colour given to the processor.[7]

For sake of simplicity, we discuss the case $C = \{*\}$, so that arcs and nodes are in fact not coloured; the introduction of colours makes no significant conceptual difference.

Formally, a *protocol* P is given by a set X of local states and by a transition function

$$\delta : X \times X^{\oplus} \to X,$$

where $X^{\oplus}$ is the set of finite multisets over X. 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).

A *global state* (for n processors, with respect to the protocol P) is a vector $\boldsymbol{x} \in X^n$. A *(synchronous) computation* of the protocol P on the network G (with n processors) is an infinite sequence of global states $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$ such that for all t and i the state x_i^{t+1} is obtained by applying δ to x_i^t and to the multiset of the x_j^t, for $j \to i$.

3.3 Behaviours, predicates and classes of networks

Classes of networks 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). Common situations studied in the literature include the knowledge of the whole network, of the underlying graph, of the number of processors, or of some other graph-theoretical property. For instance, if the only knowledge available is the number n of processors we must specify a self-stabilizing protocol working on all networks with n nodes.

Let $\mathscr{C}$ be a class of networks. A *behaviour* S for $\mathscr{C}$ on a set Y is an assignment, for every $G \in \mathscr{C}$, of a nonempty suffix-closed set of infinite sequences[8] $S_G \subseteq (Y^{n_G})^{\omega}$. A *predicate* is a behaviour all whose sequences are constant; it can be identified with an assignment, for all $G \in \mathscr{C}$, of a set $S_G \subseteq Y^{n_G}$.

For instance, the behaviour containing for each graph in $\mathscr{C}$ the sequence $\langle 0, 0, \ldots, 0\rangle, \langle 1, 1, \ldots, 1\rangle, \langle 2, 2, \ldots, 2\rangle, \ldots$ and its suffixes specifies a synchronized infinite-state clock, whereas the predicate containing all tuples $\langle b_1, b_2, \ldots, b_n\rangle$, where exactly one of the b_i's is nonzero, defines the well-known *leader election problem*. Token circulation, topology reconstruction and so on can be easily described in a similar way (see Section 8).

We say that $\mathscr{C}$ can *self-stabilize* to S iff there is a protocol P with state space $X \times Y$ such that for every network $G \in \mathscr{C}$ there is a $T \in \mathbf{N}$ with the property that for every computation

$$\langle \boldsymbol{x}^0, \boldsymbol{y}^0\rangle, \langle \boldsymbol{x}^1, \boldsymbol{y}^1\rangle, \langle \boldsymbol{x}^2, \boldsymbol{y}^2\rangle, \ldots$$

the sequence

$$\boldsymbol{y}^T, \boldsymbol{y}^{T+1}, \boldsymbol{y}^{T+2}, \ldots$$

is in S_G. The smallest such T is called the *quiescence time* of P on G, and it is denoted by T_G. The self-stabilization is *finite state* iff all computations of P on networks of $\mathscr{C}$ are ultimately periodic; this condition is equivalent to saying that all computations use a finite amount of memory (note that, however, state finiteness does not imply that X is finite, but rather that only a finite part of X is used in every computation)

[5] Since we need to manage infinite trees too, we allow the node set of a tree to be $\mathbf{N}$.

[6] We are in fact considering trees up to isomorphism (technically $\leq$ is just a preorder).

[7] Colours on the nodes act as identifiers, whereas colours on the arcs define the communication model. Choosing a suitable γ we can encode properties such as the presence of unique identifiers, distinguished outgoing/incoming links and so on [8]. Note, however, that the importance of identifiers will appear only in Section 7, where we shall exploit identifier uniqueness to improve our bounds.

[8] The set Y on which a behaviour is defined has no particular significance—it is a just a set used to build allowed sequences.

Usually, when studying self-stabilization in spite of dynamic (run-time) changes of the network topology, one strengthens the definitions above so to take into consideration this additional constraint. However, as we already mentioned, all protocols described in this paper will not use any knowledge, except for that provided by the class $\mathscr{C}$. Thus, if the topology is modified so that the resulting network still belongs to $\mathscr{C}$, we can automatically guarantee convergence to the desired behaviour (the quiescence time will of course depend on the new topology). This point will not need to be pursued hereafter.

4 Self-stabilizing view construction

A network is *anonymous* if all processors start from the same initial state and run the same program [5, 18, 22, 8, 21, 23, 10]. There is clearly a connection between the theory of anonymous networks and self-stabilization: indeed, one possible choice for the initial state is to set all processors to the same state, so, in a sense, if you can do it in a self-stabilizing way then you can do it anonymously.

The main goal of the first part of this paper is to show that *the implication above can be reversed*, that is, if there are no bounds on the amount of information exchanged by the processors, *there is essentially no advantage in being able to choose the initial local state if all local states are forced to be equal.* In other words, the worst thing an adversary can do in choosing our initial state is to set all processors to the same state.

4.1 Anonymous networks and views

A classical tool in the study of anonymous networks is the concept of *view*, introduced for bidirectional networks in [22] and extended to the directed case in [8] (to be true, the concept of view is already present in the seminal paper [5], partially disguised under the mathematical notion of *graph covering*; a finite set, called *pyramid*, of increasigly deep views decorated with states is also used in [2] to detect inconsistencies in the system). 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.

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. We will return to this property later.

4.2 Making view construction into a self-stabilizing protocol

In this section we are going to describe an infinite-state self-stabilizing protocol that allows each processor to build its own view at any desired depth. The main idea of our protocol is that a processor should build its view gathering information from its in-neighbours, as in the classical case; however, processors are interested in eliminating the "garbage" introduced by the arbitrariness of the initial state: this aim is achieved by mitigating the amount of new information introduced at each step.

Each processor i holds a tree T_i, whose value at time t we denote with T_i^t, and which is to contain a truncation of the view of i. We are interested in growing the *correct levels* of T_i^t, that is, $T_i^t \wedge \widetilde{G}^i$. To describe our protocol, we set up the following shorthands:

- We denote with S_i^t the tree obtained combining the trees T_j^t, for $j \to i$, following the definition of view. More precisely, the root of S_i^t has one child for each arc a coming into i; the corresponding subtree is T_j^t, where j is the source of a. If necessary, the resulting tree is truncated so that all maximal paths have the same length: as a consequence, $h\big(S_i^t \wedge \widetilde{G}^i\big) \geq \min_{j\to i} h\big(T_j^t \wedge \widetilde{G}^j\big) + 1$.
- We set $U_i^t = S_i^t \restriction h(T_i^t \wedge S_i^t) + 1$. Note that U_i^t is built by taking from S_i^t *one level more* than the number of levels that S_i^t shares with T_i^t.

The classical anonymous view construction algorithm would just set $T_i^{t+1} = S_i^t$. However, to control the amount of new information introduced, we will find it more fruitful to set $T_i^{t+1} = U_i^t$, that is, to truncate the new information at one level below the maximum number of levels it shares with the old one.

We remark two useful facts about U_i^t:

$$h\big(U_i^t\big) \leq \min_{j\looparrowright i} h\big(T_j^t\big) + 1 \tag{2}$$

$$h\big(S_i^t \wedge \widetilde{G}^i\big) \geq h\big(U_i^t \wedge \widetilde{G}^i\big) \geq \min_{j\looparrowright i} h\big(T_j^t \wedge \widetilde{G}^j\big) + 1. \tag{3}$$

The first statement depends trivially on the very definition of U_i^t. The second statement can be proved by noting that in case $S_i^t \leq T_i^t$, since $U_i^t = S_i^t$ we have

$$\begin{aligned} h\big(U_i^t \wedge \widetilde{G}^i\big) &= h\big(S_i^t \wedge \widetilde{G}^i\big) \\ &\geq \min_{j\to i} h\big(T_j^t \wedge \widetilde{G}^j\big) + 1 \geq \min_{j\looparrowright i} h\big(T_j^t \wedge \widetilde{G}^j\big) + 1; \end{aligned}$$

otherwise, observing that

$$\begin{aligned} h\big(S_i^t \wedge T_i^t\big) &\geq h\big(S_i^t \wedge T_i^t \wedge \widetilde{G}^i\big) \\ &= \min\big\{ h\big(S_i^t \wedge \widetilde{G}^i\big), h\big(T_i^t \wedge \widetilde{G}^i\big) \big\} \\ &\geq \min\big\{ \min_{j\to i} h\big(T_j^t \wedge \widetilde{G}^j\big) + 1, h\big(T_i^t \wedge \widetilde{G}^i\big) \big\}, \end{aligned}$$

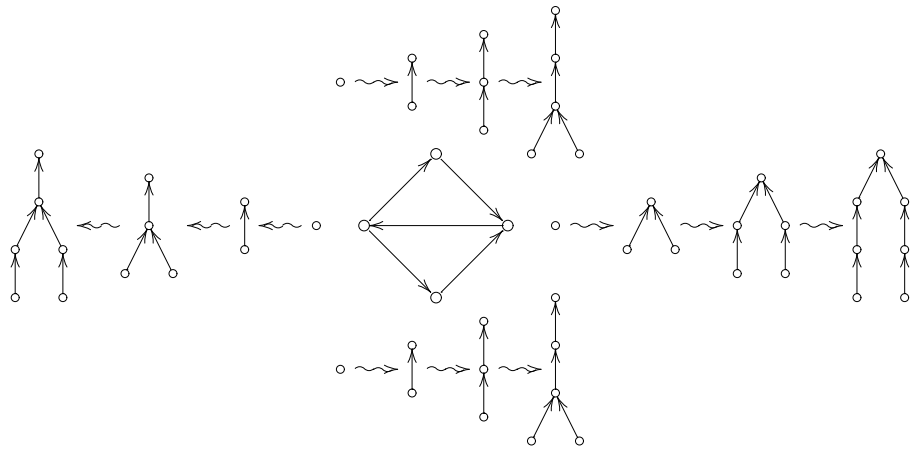


Figure 2: An example of view construction

as h preserves binary infima between comparable trees, we have

$$\begin{aligned} h\big(U_i^t \wedge \widetilde{G}^i\big) &= \min\big\{h\big(S_i^t \wedge \widetilde{G}^i\big), h\big(U_i^t\big)\big\} \\ &= \min\big\{h\big(S_i^t \wedge \widetilde{G}^i\big), h\big(S_i^t \wedge T_i^t\big) + 1\big\} \\ &\geq \min\big\{\min_{j\to i} h\big(T_j^t \wedge \widetilde{G}^j\big) + 1, \\ &\qquad \min_{j\to i} h\big(T_j^t \wedge \widetilde{G}^j\big) + 2, \\ &\qquad h\big(T_i^t \wedge \widetilde{G}^i\big) + 1\big\} \\ &= \min_{j\looparrowright i} h\big(T_j^t \wedge \widetilde{G}^j\big) + 1, \end{aligned}$$

using (1) and the definition of U_i^t (assuming $S_i^t \not\leq T_i^t$).

4.3 The infinite-state protocol

The protocol we use in the infinite-state case is extremely simple:

$$\boxed{T_i^{t+1} = U_i^t} \tag{4}$$

Informally, each processor compares the candidate new view with its own current view, finds the greatest k such that the two trees truncated at depth k are the same, and then takes from the candidate new view just $k+1$ levels. Thus, each processor trusts the trees passed by its in-neighbours, but in a controlled way. A graphical display of the updated rule is given in Figure 3.

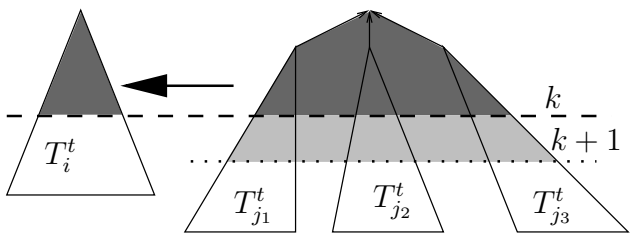


Figure 3: The infinite-state protocol update rule.

Inequality (3) will be the base for the proof of self-stabilization, as it establishes that the number of correct levels of T_i^t (i.e., the height of $T_i^t \wedge \widetilde{G}^i$) grows with time. Indeed, it is immediate to prove the following

Lemma 1 For all $t \geq 0$ we have $\min_i h\big(T_i^{t+1} \wedge \widetilde{G}^i\big) \geq \min_i h\big(T_i^t \wedge \widetilde{G}^i\big) + 1$.

This lemma claims that the *minimum* number of correct levels grows with time. But this growth is just half of what we need: it is also necessary to show that at some time all trees will become *entirely* correct. We will obtain this result proving that some trees immediately become short (and correct), and that this property is inherited by their out-neighbours; thus, correctness spreads to the whole network in D steps. Since we shall use this proof technique very frequently, we formalize it through the following

Proposition 1 Let $\boldsymbol{x}^0, \boldsymbol{x}^1, \ldots$ be a computation of some protocol on G, $\bar{t} \in \mathbf{N}$ and $f : X \to \mathbf{N}$ be a function such that for all $t \geq \bar{t}$ we have $f\big(x_i^{t+1}\big) \leq \min_{j\looparrowright i} f\big(x_j^t\big) + 1$. Then $f\big(x_j^t\big) \leq \min_i f\big(x_i^{\bar{t}}\big) + t - \bar{t}$ for all j and $t \geq \bar{t} + D$.

Proof. Let l be any processor such that $f\big(x_l^{\bar{t}}\big) = \min_i f\big(x_i^{\bar{t}}\big)$. We prove by induction on d that for all j such that $d(l,j) \leq d$ and for all $t \geq \bar{t} + d$ we have $f\big(x_j^t\big) \leq \min_i f\big(x_i^{\bar{t}}\big) + t - \bar{t}$. The base case ($j = l$) is a straightforward induction on t. On the other hand, if $d(l,j) = d+1$, then there is a $j' \to j$ at distance d from l, so by induction hypothesis $f\big(x_{j'}^t\big) \leq \min_i f\big(x_i^{\bar{t}}\big) + t - \bar{t}$ for all $t \geq \bar{t} + d$, and finally, for $t \geq \bar{t} + d + 1$

$$\begin{aligned} f\big(x_j^t\big) \leq \min_{i\looparrowright j} f\big(x_i^{t-1}\big) + 1 \leq f\big(x_{j'}^{t-1}\big) + 1 \\ \leq \min_i f\big(x_i^{\bar{t}}\big) + t - \bar{t}. \end{aligned}$$ ∎

We now come to the main proof of this section. The basic idea is that, due to the update rule we adopted, the processor with the minimum number of correct levels (i.e., with the most scrambled information) will become correct at the first step.

Theorem 1 There exists an integer c such that after $t > D$ steps of protocol (4) all processors possess a correct truncation of their own view of depth $t + c$ (i.e., $T_i^t = \widetilde{G}^i \restriction (t+c)$).

Proof. Let $c = \min_i h\big(T_i^0 \wedge \widetilde{G}^i\big)$ be the minimum correctness level, and j be a node with minimum correctness (i.e., such that $h\big(T_j^0 \wedge \widetilde{G}^j\big) = c$). Since $h\big(S_j^0 \wedge \widetilde{G}^j\big) > c$ by (3), necessarily $h\big(S_j^0 \wedge T_j^0\big) = c$ (otherwise T_j^0 would share more than c levels with $\widetilde{G}^i$); thus, after the first step, $h\big(T_j^1\big) = c + 1$.

Since the tree height function h satisfies the hypotheses of Proposition 1, and T_j^1 has height $c + 1$, we have $h\big(T_i^t\big) \leq c + t$ for all i and all $t > D$. On the other hand, $c + t \leq h\big(T_i^t \wedge \widetilde{G}^i\big) \leq h\big(T_i^t\big)$ by Lemma 1, so for $t > D$ we have $h\big(T_i^t\big) = c + t$ and $T_i^t \leq \widetilde{G}^i$ for all i, and the thesis follows. ∎

Table 1 shows the first steps of an execution of the algorithm on the network G of Figure 2: the table contains the value of T_i^t at each step. Observe that at step 3 all truncated views are correct. Please ignore for the time being the last column.

5 Graph-theoretical characterizations of attainable behaviours

In the previous section we have shown how to build processor views in a self-stabilizing way. Thus, if we already know that our desired behaviour just depends on (a truncation of) the view, we can realize it *via* protocol (4).

On the other hand, the theory of anonymous networks tells us that no protocol can attain a behaviour that does not depend exclusively on the view, as we remarked informally in the previous sections. Thus, in principle we have characterized mathematically the attainable behaviours.

However, it is clear that this knowledge does not help us much in solving automatically the problem stated in Section 2. More importantly, we have no clue about how to turn our protocol into a finite-state one. This is the reason why we are going to study in much deeper detail the constraints on the behaviours of a self-stabilizing network.

5.1 Graph fibrations and the Lifting lemma

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.[9]

To express our combinatorial characterization of attainable behaviours, we shall exploit the notion of *graph fibration* [9]. 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 [5] and the concept of similarity of processors introduced in [18].

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 4 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

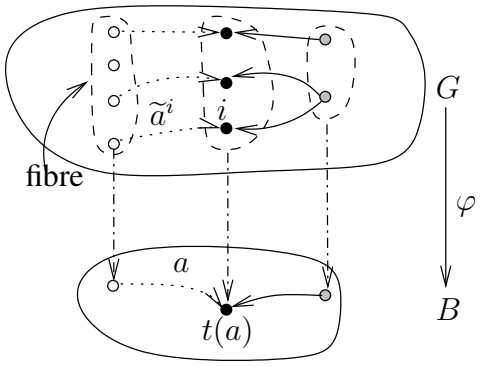


Figure 4: A fibration.

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, $(\boldsymbol{x}^\varphi)_i = x_{\varphi(i)}$. 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 2 (Lifting lemma) Let $\varphi : G \to B$ be a fibration. Then, for every protocol P and every computation $\boldsymbol{x}^0, \boldsymbol{x}^1, \ldots$ of P on B, $\big(\boldsymbol{x}^0\big)^\varphi, \big(\boldsymbol{x}^1\big)^\varphi, \ldots$ is a computation of P on G.

Proof. Whenever a network is fibred onto another one, and processors in the same fibre are in the same state as the processor they are mapped to by the fibration, all processors in the same fibre must behave identically. This happens because the local isomorphism property says that for any two processors i and j in the same fibre, there is an arc terminating at j and starting from a processor in the same fibre as k; moreover, this association is a bijection between the arcs entering in i and j. Thus, the sequence of local states of all processors in a fibre of φ are exactly the same, and they are also equal to the sequence of states of the processor they lie over. ∎

[9] This is the only definition in this paper that should be modified to handle the coloured case: one has to consider only morphisms that preserve colours on the nodes and on the arcs. All the graph-theoretical ideas we introduce adapt smoothly to this condition.

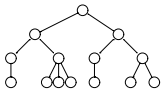
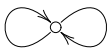
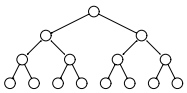
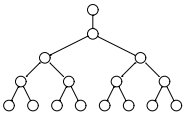
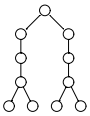
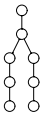
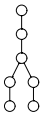
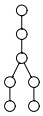
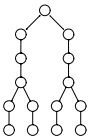
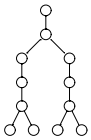
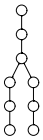
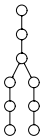
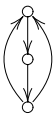
t	T_1^t	T_2^t	T_3^t	T_4^t	$\mathscr{B}(T_4^t)$
0					
1					
2					
3					undefined
4					undefined
5					

Table 1: An execution of the infinite-state algorithm on the network G of Figure 2.

The above lemma suggests that if a network can be "collapsed" by a fibration onto another one, then there are certain constraints on the behaviours it can self-stabilize to. More precisely, we can compute a sequence of lifted global states of G by computing a sequence of global states of B, and then lift the result. In other words, the computation of B "resumes" in a more compact way the computation of G. Since a possible initial global state for G is the one in which all processor have the same local state, we cannot hope to self-stabilize to a behaviour that is not a lifting of a behaviour of B.

This consideration may seem trivial when applied to a single network, but it becomes more tricky when a whole class is involved. Indeed, if *two* different networks G and H of the class are fibred over a common base B, we must find a *single* behaviour of B that, once lifted to G and H, will give in *both* cases a desired behaviour.

The above observation is already sufficient to show that there is no self-stabilizing protocol for election for the set of networks given in Figure 1. Indeed, the two bottom networks have a common base, as shown by the labellings displayed in Figure 5 (the reader can easily check that the labellings are indeed the node components of two fibrations: for instance, every node with label 4 has two incoming arcs, one from a node with label 4 and one from a node with label 3). We have already noticed that the only

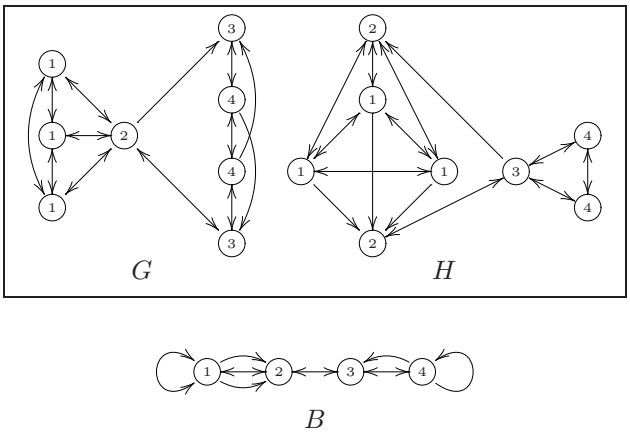


Figure 5: Two networks with a common base.

behaviours we can attain on G and H are liftings of the behaviours of B. This means that *election is impossible on the class* $\{G, H\}$. Indeed, we can only choose between four predicates on B, that is, $\langle 1,0,0,0\rangle$, $\langle 0,1,0,0\rangle$, $\langle 0,0,1,0\rangle$ and $\langle 0,0,0,1\rangle$, and all of them, when lifted, do not give rise to a unique leader either on G or on H. The main point to notice is that, as we shall see, election on G *or* on H *is* possible. In other words, if the processors know whether they live in G or H they can do self-stabilizing election, but if this knowledge is denied, then self-stabilizing election becomes impossible (and, of course, is *a fortiori* impossible on the whole class depicted in Figure 1).

The example shows that it is essential to find out which networks in a class are fibred onto a common base. The interesting features of the synchronous model is that the search can be reduced to a particular kind of fibration, that collapses a network as much as possible.

5.2 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[10].

Theorem 2 ([9]) The following properties hold:

1. For every graph G there is exactly one fibration prime graph $\hat{G}$, called the *minimum base* of G, such that G is fibred onto $\hat{G}$.
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 [9], similar to the one used for finite-state automata 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. 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 numbering the nodes.

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 2, so we denote with μ_G one of them when the map on the arcs is not relevant. For instance, in Figure 5 the graph B is the minimum base of both G and H, that is, $\hat{G} \cong \hat{H} \cong B$.

The previous theorem highlights the deep link between fibrations and views: two processors have the same view if and only if[11] they lie in the same fibre of μ_G. 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 two following theorems:

Theorem 3 ([9]) Let G be a strongly connected graph and B a fibration prime graph with minimum number of nodes satisfying $h\big(\widetilde{G}^i \wedge \widetilde{B}^j\big) \geq n_G + D_G$ for some $i \in N_G$ and $j \in N_B$: then $B \cong \hat{G}$.

In other words, if we have a network G and a candidate minimum base B for G, we can check whether B is the minimum base of G by finding a processor of G and a processor of B that share at least $n_G + D_G$ level of their views, and by checking that no network

[10]Note that in the coloured case, primeness does not depend only on the structure of a graph, but also on its colours.

[11]In fact, the connection is much deeper, as the view of a processor i in a network G is the only in-tree fibred over G whose root is mapped to i.

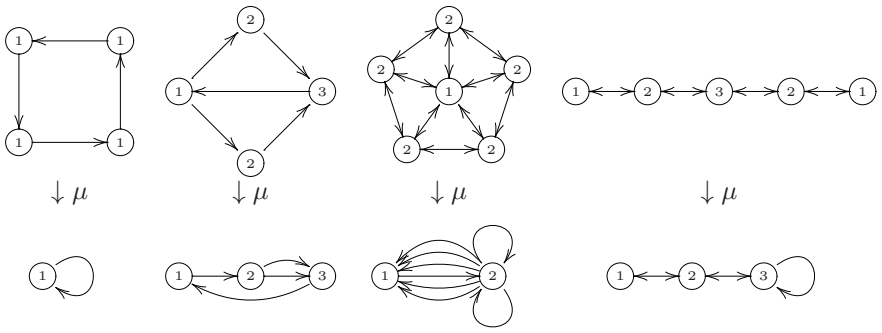


Figure 6: Some examples of minimal fibrations and minimum bases.

with less nodes than B has this property. This fact will enable us to recover the minimum base from a sufficiently deep truncation of a view, by suitably enumerating all fibration prime graphs.

Theorem 4 (Norris [20]) $\widetilde{G}^i \cong \widetilde{G}^j$ iff $h\big(\widetilde{G}^i \wedge \widetilde{G}^j\big) \geq n_G - 1$.

Theorem 4 tells us that to distinguish different views of a network G we just have to check isomorphism of the first $n_G - 1$ levels. Observe that if G is fibred over B, then the view of a processor in G is the same as the view of the processor of B on which it is mapped; hence, a processor can decide from a sufficiently deep view the node of the minimum base it is mapped to by a minimal fibration.

5.3 Deducing the minimum base from the view

An immediate consequence of the Lifting lemma is that the only data we need to self-stabilize to a behaviour are the minimum base of the network and a minimal fibration, plus a synchronized clock. Note that we really need just the node component of the minimal fibration, and that clocks are of course unnecessary in the case of predicates.

This situation is similar to the typical scenario on an anonymous network [8]: all processors build a sufficiently deep truncation of their view, and then apply Theorem 4 and 3. Since we want our protocol to be able to run without any knowledge (for instance, without knowing a bound on the number of processors), we must provide a way to deduce the minimum base from truncated views. To obtain an optimal quiescence time, we also want to make this deduction as early as possible.

To this purpose, we introduce a partial function $\mathscr{B}$ that allows processors to extract from their truncated view a candidate for the minimum base of the network that the protocol is running on, without assuming any other knowledge.

More precisely, $\mathscr{B}$ assigns to a tree T a fibration prime graph in such a way that T is a prefix of a view of $\mathscr{B}(T)$. We will set up $\mathscr{B}$ in such a way that when T does not contain enough information, $\mathscr{B}(T)$ is undefined. Formally:

Definition 2 Let T be a tree. If the set

$$\Big\{ G \;\Big|\; G \text{ is fibration prime and } h(T) \geq n_G + D_G \text{ and } \exists i \in N_G \text{ such that } T \leq \widetilde{G}^i \Big\}$$

is nonempty, then $\mathscr{B}(T)$ is the graph in the set above having the least number of nodes (such a graph is unique up to isomorphism by Theorem 3, and we choose a fixed representative for each isomorphism class). Otherwise, $\mathscr{B}(T)$ is undefined.

Note that $\mathscr{B}$ is easily computable by enumeration when T is finite. It is immediate to prove (using Theorem 3) that when $\mathscr{B}$ is fed with a sufficiently deep truncation of a view, it returns the correct minimum base:

Proposition 2 Let G be a graph. For every node i of G and every $k \geq n_{\hat{G}} + D_{\hat{G}}$ we have $\mathscr{B}\big(\widetilde{G}^i \restriction k\big) \cong \hat{G}$.

In particular, the above statement is true for $k \geq n_G + D_G$. The other fundamental property of $\mathscr{B}$ we shall need is described in the following

Proposition 3 If $\mathscr{B}(T)$ is defined, there is a unique node i of $\mathscr{B}(T)$ such that $T \leq \widetilde{\mathscr{B}(T)}^i$. Moreover, for every tree U such that $h(T \wedge U) \geq n_{\mathscr{B}(T)} - 1$, if $\mathscr{B}(T) \cong \mathscr{B}(U)$ then $U \leq \widetilde{\mathscr{B}(T)}^i$.

Proof. Let $B = \mathscr{B}(T)$. Existence follows from the definition of $\mathscr{B}$. Moreover, if there is a j such that $T \leq \widetilde{B}^j$, since $h(T) \geq n_B - 1$, we have $h(\widetilde{B}^i \wedge \widetilde{B}^j) \geq n_B - 1$, and by Theorem 4 $i = j$. Finally, there is a node j of B such that $U \leq \widetilde{B}^j$; since $h(U \wedge T) \geq n_B - 1$, we have again $j = i$. ∎

As a general rule, whenever we write $\mathscr{B}(T) \cong \mathscr{B}(U)$, we understand that $\mathscr{B}$ is defined on both T and U (in this case, by definition of $\mathscr{B}$ the isomorphism is actually an identity). In Table 2 we show the only trees of height at most three and indegree at most two on which $\mathscr{B}$ is defined, and the corresponding output value (the arcs of the trees are of course oriented towards the root). Moreover, the invite the reader to look back at Table 1: the last column shows the value of $\mathscr{B}$ on the view of the last processor.

5.4 The characterization theorem

Theorem 5 Let $\mathscr{C}$ be a class of networks and S a behaviour on Y. Then, the following two conditions are equivalent:

- there is a protocol that self-stabilizes to S;

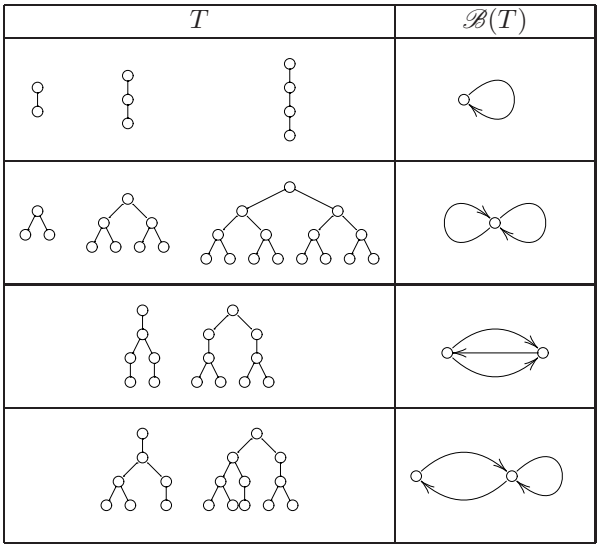


Table 2: Some values of $\mathscr{B}$ on small trees.

- for every fibration prime graph B there is a sequence $\boldsymbol{y}_B^0$, $\boldsymbol{y}_B^1, \ldots \in (Y^{n_B})^\omega$ such that for all $G \in \mathscr{C}$

$$\left(\boldsymbol{y}_{\hat{G}}^{T_G}\right)^{\mu_G}, \left(\boldsymbol{y}_{\hat{G}}^{T_G+1}\right)^{\mu_G}, \cdots \in S_G$$

  for some $T_G \in \mathbf{N}$.

If the above holds, $\mathscr{C}$ can self-stabilize to S with quiescence time at most $\max\{n_G + D_G, T_G\}$.

Proof. We just need to prove the positive part, the negative part being a trivial application of the Lifting lemma. All processors build their view using protocol (4), and at each step apply the function $\mathscr{B}$ to obtain a guess B on the minimum base (and on the node j of the minimum base to which they would be mapped to); then, they set the Y-part of their state space to the j-th component of $\boldsymbol{y}_B^h$, where h is the current height of the view (i.e., the processors use the current view height as a synchronized clock). As a result, the Y-part of the sequence of global states will be in S. The quiescence time is a consequence of Theorem 1 and Proposition 2. ∎

Theorem 5 is twofold: on one side, it gives us a necessary and sufficient condition to discover whether a class can self-stabilize to a given behaviour; on the other side, its proof is based on a *universal self-stabilizing protocol* that works whenever the behaviour is attainable. Note that there is a part of the protocol that depends on the behaviour (the sequence associated to a minimum base), but this part is the same for all processors, and does not contain any "control logic" (it is used essentially as an oracle—different oracles will give rise to all possible attainable behaviours).

The characterization implied by statement of Theorem 5 may seem to be daunting, but it really boils down to the following steps:

1. Find out the minimum bases of the networks in $\mathscr{C}$.
2. For each of them, try to find out a sequence of global Y-states satisfying the statement of the theorem (i.e., the sequence, once lifted to a network of $\mathscr{C}$ should represent a desired behaviour).
3. If the previous step is not possible, then the behaviour is not attainable; otherwise, the chosen sequences can be fed to the universal protocol, giving rise to a self-stabilizing protocol for the behaviour.

Of course, when $\mathscr{C}$ and Y are finite the above steps give an effective procedure. Otherwise, Theorem 5 is still a powerful tool for proving possibility and impossibility results, as exemplified in Section 8.

Let us apply the technique to the example given in Figure 1. Some standard computations show that there are just two minimum bases for the networks: one was given in Figure 5, and the other one is presented in Figure 7. Looking at Figure 7 we notice

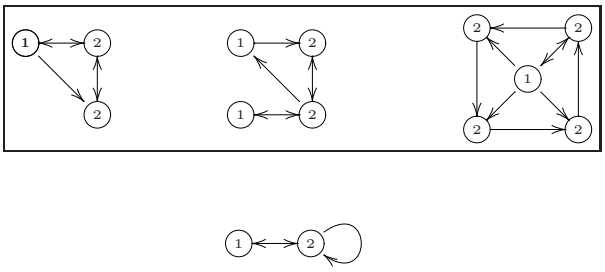


Figure 7: The remaining networks and their minimum base.

that the second network is an unsurmountable obstacle to election: any predicate on the base ($\langle 0, 1\rangle$ or $\langle 1, 0\rangle$), once lifted to this network, gives an incorrect behaviour. On the other hand, the other two networks do not give problems, since there is always exactly one processor that is mapped to processor 1 of the base. We conclude that self-stabilizing election is not possible in the class shown in Figure 1; nonetheless, by eliminating the four-node network of Figure 7 and one of the two bigger networks of Figure 5 self-stabilizing election becomes possible. The reader can easily compute the two sequences used by the universal protocol.

A simple corollary of Theorem 5 gives us the attainable predicates:

Corollary 1 Let $\mathscr{C}$ be a class of networks and S a predicate on Y. Then, the following two conditions are equivalent:

- there is a protocol that self-stabilizes to S;
- for every fibration prime graph B there is a $\boldsymbol{y}_B \in Y^{n_B}$ such that $\left(\boldsymbol{y}_{\hat{G}}\right)^{\mu_G} \in S_G$ for all $G \in \mathscr{C}$.

If the above holds, $\mathscr{C}$ can self-stabilize to S with quiescence time at most $n_G + D_G$.

5.5 A lower bound

We now show that our bound on quiescence time is tight. We base our proof on the fact that the bound $n_G + D_G$ of Theorem 3 is tight: in [9] the authors proved that the fibration prime graphs $G_{n,D}$ and $H_{n,D}$ of Figure 8, which have n nodes and diameter D, have the property that $\widetilde{G_{n,D}}^1$ and $\widetilde{H_{n,D}}^1$ are isomorphic up to level $n + D - 1$, but not up to level $n + D$ (the difference between the two families is given by the positioning of the dotted arc). In other words, processors living on such networks need a long time

($n + D$ steps) to understand which kind of network they belong to. Let now $\mathscr{C}$ be the class of networks $G_{n,D}$ and $H_{n,D}$ described in Figure 8 and $\mathscr{C}_N$ the class of networks of $\mathscr{C}$ with at most N nodes. From the theory of anonymous networks we know that

Lemma 3 Processor 1 of network $G_{n,D}$ and processor 1 of network $H_{n,D}$ have the same state for $n + D - 1$ steps of every anonymous computation (i.e., every computation in which all processors start from the same state).

This is immediate, as the state of a processor at time t depends only on its view truncated at depth t.

Theorem 6 Every protocol that self-stabilizes $\mathscr{C}_N$ to the predicate

$$MB_G = \{\langle\langle \hat{G}, \mu_G(1)\rangle, \langle \hat{G}, \mu_G(2)\rangle, \ldots, \langle \hat{G}, \mu_G(n_G)\rangle\rangle\},$$

has quiescence time at least $n + D$ for at least $|\mathscr{C}_N|/2$ networks. Note that essentially predicate MB forces each processor to compute the network topology and its own identity in such network (since all networks in $\mathscr{C}$ are fibration prime).

Proof. If a network with underlying graph $G_{n,D}$ stabilizes before $n + D$ steps, by Lemma 3 the network $H_{n,D}$ cannot stabilize before $n + D$ steps. ∎

Corollary 2 Every protocol that self-stabilizes $\mathscr{C}$ (or a larger class—in particular, the class of fibration prime graphs) to the predicate MB has quiescence time at least $n + D$ for infinite networks.

6 Finite-state universal self-stabilization

The previous sections should have made clear that computing the minimum base and the node to which a processor is mapped, that is, the predicate

$$MB_G = \{\langle\langle \hat{G}, \mu_G(1)\rangle, \langle \hat{G}, \mu_G(2)\rangle, \ldots, \langle \hat{G}, \mu_G(n_G)\rangle\rangle\},$$

that we defined in Section 5.5, is all we need to self-stabilize to an attainable predicate. Moreover, a synchronized clock is necessary to self-stabilize to a generic behaviour.

Note that if we require state finiteness, then every computation must be ultimately periodic, and, similarly, all behaviours must be ultimately periodic. Thus, a finite-state modular clock would be sufficient.

Unfortunately, the protocol we discussed uses an unbounded number of states (the views are always growing). This is not a problem if, for instance, a bound on the number of processor is known: in this case, we can safely truncate all views at, say, twice the number of processors, and obtain a correct guess for the minimum base. The resulting protocol would be finite state and, coupled with the synchronized clock described in Section 6.1, would give rise to a universal finite-state self-stabilizing protocol.

Clearly, there are some behaviours, such as a synchronized non-wrapping clock, that requires necessarily infinite state, but important problems (such as election) should have in principle a finite-state self-stabilizing solution.

To overcome this problem, we introduce a *guess* $m_i > 0$ for each processor i, which is used to keep the height of T_i under control. We shall set up our protocol in such a way that if m_i is too small, some locally checkable conditions will become true, forcing an update of the guess. A certain number of such local checks and corrections will be necessary, and this fact will increase the quiescence time. Thus, we parameterize our protocol with respect to an update function g, and show that we can make the loss in quiescence time arbitrarily small by choosing an update function g growing quickly enough. There is of course a tradeoff with the space used by the protocol.

Let $g : \mathbf{N} \to \mathbf{N}$ be a strictly increasing and strictly inflationary function (i.e., a function such that $m < g(m) < g(m + 1)$ for all $m \in \mathbf{N}$). We use g^k to denote the k-th iterate of g, and define

$$g^*(m) = \min\{k \mid g^k(0) \geq m\}.$$

It is rather easy to show that $g^*(m) \leq m$, because $g^k(m) \geq m + k$, that $g^k(m) \geq g^k(0) + m$ and that, choosing g suitably, we can make g^* grow as slowly as desired.

The finite-state protocol self-stabilizing to MB is the following one:

$$\begin{aligned} T_i^{t+1} &= U_i^t \restriction m_i^t \\ m_i^{t+1} &= \begin{cases} \max\limits_{j \looparrowright i} m_j^t & \text{if } \mathscr{B}\big(T_i^t\big) \cong \mathscr{B}\big(U_i^t\big) \\ & \cong \mathscr{B}\big(T_j^t\big) \text{ for all } j \to i \\ g\big(\max\limits_{j \looparrowright i} m_j^t\big) & \text{otherwise.} \end{cases} \end{aligned} \tag{5}$$

The intuition behind the protocol is that we proceed as in the infinite-state case, but we keep just m_i levels of our candidate view. At each step all processors check that their current idea about the minimum base is the same for all in-neighbours, and moreover that the idea would not be changed by the next update. If this does not happen, then the guess m_i is increased using the function g.

The proof of correctness of the protocol is rather complex. The most delicate part consists in showing that if all processors have the same idea about the minimum base, and they are not going to change it at the next step, then the idea is the right one.

We now prove a number of lemmata that will lead us to the main theorem. First of all, we show that the number of correct levels of a tree increases at each step (as in Lemma 1), but in a bounded way (we cannot expect it to grow at least by one at each step, because U_i^t is now truncated on the basis of the guess).

Lemma 4 Let $m = \min_i m_i^{\bar{t}}$ and $c = \min_i h\big(T_i^{\bar{t}} \wedge \widetilde{G}^i\big)$. Then, for all $t \geq \bar{t}$,

(a). $h\big(T_i^t \wedge \widetilde{G}^i\big) \geq \min\{c + t - \bar{t}, m\}$;

(b). $h\big(T_i^t \wedge \widetilde{G}^i\big) \geq \min\big\{c + t - \bar{t}, h\big(T_i^t\big)\big\}$;

(c). if $\bar{t} > 0$, $\min_i h\big(T_i^t \wedge \widetilde{G}^i\big) \geq \min_i h\big(T_i^{\bar{t}} \wedge \widetilde{G}^i\big)$.

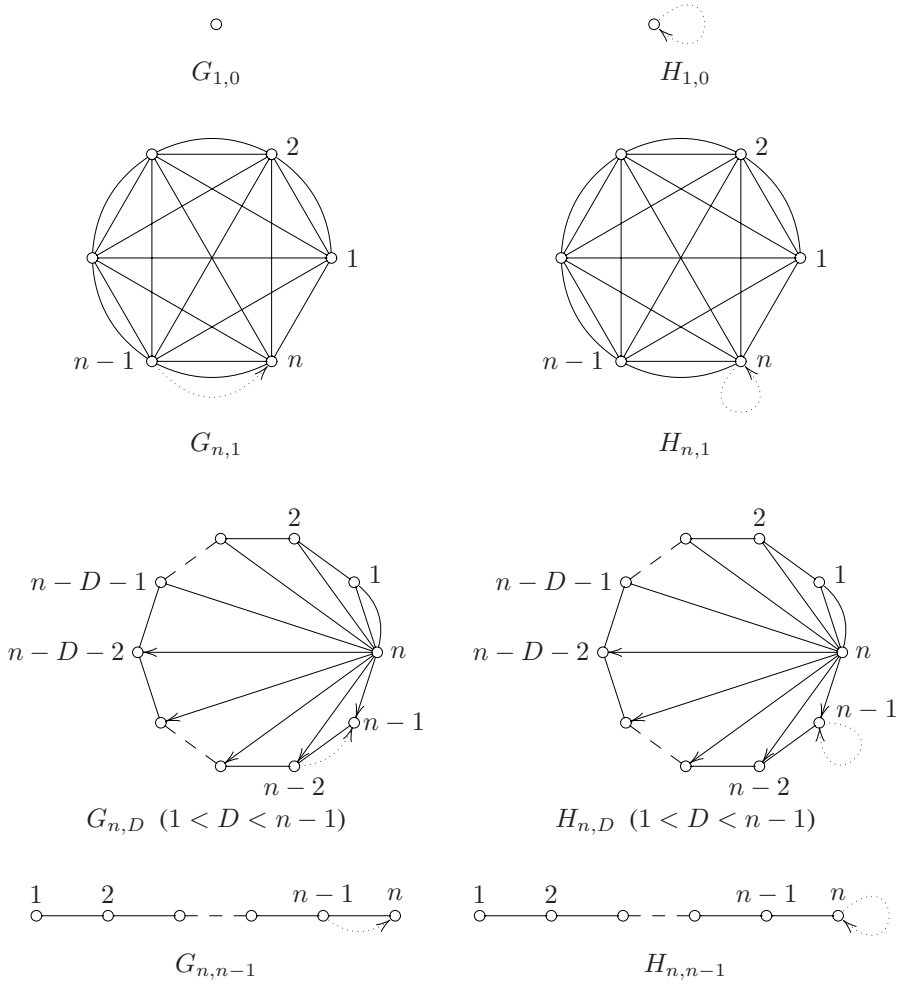


Figure 8: Networks with similar views.

Proof. (a). The case $t = \bar{t}$ is immediate. For $t \geq \bar{t}$,

$$\begin{aligned} h\big(T_i^{t+1} \wedge \widetilde{G}^i\big) &= \min\big\{ h\big(U_i^t \wedge \widetilde{G}^i\big), m_i^t \big\} \\ &\geq \min\big\{ \min_{j \looparrowright i} h\big(T_j^t \wedge \widetilde{G}^j\big) + 1, m \big\} \\ &\geq \min\{ c + t - \bar{t} + 1, m + 1, m \} \\ &= \min\{ c + t - \bar{t} + 1, m \}, \end{aligned}$$

using (1) and (3).

(b). The case $t = \bar{t}$ is immediate, as $h\big(T_i^{\bar{t}} \wedge \widetilde{G}^i\big) \geq c = \min\big\{c, h\big(T_i^{\bar{t}}\big)\big\}$. For $t \geq \bar{t}$,

$$\begin{aligned} h\big(T_i^{t+1} \wedge \widetilde{G}^i\big) &= \min\big\{ h\big(U_i^t \wedge \widetilde{G}^i\big), h\big(T_i^{t+1}\big) \big\} \\ &\geq \min\big\{ \min_{j \looparrowright i} h\big(T_j^t \wedge \widetilde{G}^j\big) + 1, h\big(T_i^{t+1}\big) \big\} \\ &\geq \min\big\{ c + t - \bar{t} + 1, \min_{j \looparrowright i} h\big(T_j^t\big) + 1, \\ &\qquad h\big(T_i^{t+1}\big) \big\} \\ &= \min\big\{ c + t - \bar{t} + 1, h\big(T_i^{t+1}\big) \big\}, \end{aligned}$$

using again (1), (3) and (2).

(c). Note that certainly $m_i^{\bar{t}} \geq m_i^{\bar{t}-1} \geq h\big(T_i^{\bar{t}}\big) \geq h\big(T_i^{\bar{t}} \wedge \widetilde{G}^i\big)$. Thus, by (a)

$$\begin{aligned} &\min_i h\big(T_i^t \wedge \widetilde{G}^i\big) \\ &\qquad \geq \min\big\{ \min_i h\big(T_i^{\bar{t}} \wedge \widetilde{G}^i\big) + t - \bar{t}, \min_i m_i^{\bar{t}} \big\} \\ &\qquad\qquad \geq \min_i h\big(T_i^{\bar{t}} \wedge \widetilde{G}^i\big). \blacksquare \end{aligned}$$

The above theorem guarantees that, under suitable conditions, the number of correct levels will increase. However, we must also show that at some time trees will be correct (i.e., $T_i^t \leq \widetilde{G}^i$). As in the proof of Theorem 1, we obtain this result by showing that trees with minimum correctness become correct at the first step, and that correctness is inherited by out-neighbours.

Lemma 5 For all $t > D$ we have $T_i^t \leq \widetilde{G}^i$ for all i.

Proof. Let $c = \min_i h\big(T_i^0 \wedge \widetilde{G}^i\big)$ be the minimum correctness level, and j be a node with minimum correctness. Since after the first step

$$h\big(T_j^1\big) = c + 1 \geq h\big(T_j^1 \wedge \widetilde{G}^j\big) \geq \min\big\{ c + 1, h\big(T_j^1\big) \big\}$$

by Lemma 4.(b), we can apply Proposition 1 (using the tree height as f), and obtain $h\big(T_i^t\big) \leq c + t$ for all i and all $t > D$. On the other hand, $h\big(T_i^t \wedge \widetilde{G}^i\big) \geq \min\big\{ c + t, h\big(T_i^t\big) \big\} = h\big(T_i^t\big)$, again by Lemma 4.(b), and the thesis follows. $\blacksquare$

We now prove that if no processor applies g, then we are in a state satisfying the predicate. Said otherwise, the local conditions that we check guarantee that globally all processors are computing the minimum base correctly.

Lemma 6 If $\mathscr{B}\big(T_i^t\big) \cong \mathscr{B}\big(U_i^t\big) \cong \mathscr{B}\big(T_j^t\big)$ for all i and all $j \to i$, then $\mathscr{B}\big(T_i^t\big) \cong \hat{G}$ and $T_i^t \leq \widetilde{G}^i$ for all i.

Proof. We prove the statement by building a fibration $\varphi : G \to B$, B being the common value of all $\mathscr{B}\big(T_i\big)$ (we drop the superscript t throughout the proof); φ will of course be minimal. For all i, let us define $\varphi(i)$ as the unique (by Proposition 3) node of B such that $T_i \leq \widetilde{B}^{\varphi(i)}$. Since $h(U_i \wedge T_i) \geq h(U_i) - 1 \geq n_B - 1$, by Proposition 3 we have also $U_i \leq \widetilde{B}^{\varphi(i)}$.

Let $\upsilon : \widetilde{B}^{\varphi(i)} \to B$ be the fibration mapping each arc of $\widetilde{B}^{\varphi(i)}$ to the corresponding arc of B (recall that the nodes of $\widetilde{B}^{\varphi(i)}$ are paths into $\varphi(i)$, and that each arc of $\widetilde{B}^{\varphi(i)}$ arises from a unique arc of B). We define φ on the arcs of G as follows: each arc $j \xrightarrow{a} i$ of G corresponds uniquely to an arc a' coming into the root of $U_i \leq \widetilde{B}^{\varphi(i)}$, so if we identify U_i with $\widetilde{B}^{\varphi(i)} \restriction h(U_i)$ then we can map a to $\upsilon(a')$.

To show that φ is a graph morphism, we must prove it commutes with the source and target maps. The latter case is immediate, noting that

$$t(\varphi(a)) = t\big(\upsilon(a')\big) = \upsilon(t(a')) = \varphi(i) = \varphi(t(a)).$$

For the former case, we have

$$s(\varphi(a)) = s\big(\upsilon(a')\big) = \upsilon(s(a')).$$

Since T_j is an extension of a first-level subtree V of U_i, we have $V \leq T_j \leq \widetilde{B}^{\varphi(j)}$. Moreover, since $h(V) = h(U_i) - 1 \geq n_B - 1$, if for some k we have $V \leq \widetilde{B}^k$ then necessarily $k = \varphi(j)$. But $V \leq \widetilde{B}^{\upsilon(s(a'))}$ by definition of view, so $s(\varphi(a)) = \varphi(s(a))$.

The fact that φ is a fibration follows easily from its definition on the arcs, so we obtain $B \cong \hat{G}$ and, consequently, $T_i \leq \widetilde{B}^{\varphi(i)} \cong \widetilde{G}^i$. $\blacksquare$

Of course, we need to show that the above conditions, once satisfied, remain true forever. This is the purpose of the next

Lemma 7 If there is a $\bar{t} > 0$ such that $\mathscr{B}\big(T_i^{\bar{t}}\big) \cong \hat{G}$ and $T_i^{\bar{t}} \leq \widetilde{G}^i$ for all i, then these properties are true for all $t \geq \bar{t}$.

Proof. Since $h\big(T_i^{\bar{t}}\big) \geq n_{\hat{G}} + D_{\hat{G}}$ for all i, we have

$$\begin{aligned} h\big(T_i^{\bar{t}+1}\big) &\geq \min\big\{ n_{\hat{G}} + D_{\hat{G}} + 1, m_i^{\bar{t}} \big\} \\ &\geq \min\big\{ n_{\hat{G}} + D_{\hat{G}} + 1, h\big(T_i^{\bar{t}}\big) \big\} \geq n_{\hat{G}} + D_{\hat{G}}. \end{aligned}$$

By definition $T_i^{\bar{t}+1} \leq U_i^{\bar{t}} \leq \widetilde{G}^i$, so by Proposition 2 $\mathscr{B}\big(T_i^{\bar{t}+1}\big) \cong \hat{G}$. $\blacksquare$

The above proposition shows that if Protocol (5) is self-stabilizing, it is certainly finite state, as upon stabilization g is never applied, so in finite time all processors enter a fixed point (U_i^t is always truncated at height m_i^t, and thus T_i^t does not grow anymore).

Finally, we have to give bounds for quiescence time. To this purpose, we firstly show that if stabilization has not been reached yet, g has been applied often enough:

Lemma 8 Let $m = \min_i m_i^0$. For all $k \in \mathbf{N}$, letting $t_k = kD + g^k(m)$, one of the two following statements is true:

(a). for all i we have $\mathscr{B}\big(T_i^{t_k}\big) \cong \hat{G}$ and $T_i^{t_k} \leq \widetilde{G}^i$;

(b). $\max_i m_i^{t_k} \geq g^{k+1}(m)$ and, for all $t_k + D \leq t \leq t_{k+1}$, we have $h\big(T_i^t \wedge \widetilde{G}^i\big) \geq t - (k+1)D$.

Proof. Since statement (a) is stable (Lemma 7), we can prove our claim by induction assuming that (b) is true and (a) is false. For $k = 0$ we have $\max_i m_i^m \geq g(m)$, as at least one processor applies g if statement (a) is not true (recall that $m > 0$). Thus, $\min_i m_i^{m+D} \geq g(m)$, so for $m + D \leq t \leq D + g(m)$ we have

$$h\big(T_i^t \wedge \widetilde{G}^i\big) \geq \min\{m + t - m - D, g(m)\} = t - D$$

using Lemma 4.(a) first with $\bar{t} = m + D$ and then with $\bar{t} = 0$.

For the general case, note that certainly $\min_i m_i^{t_k+D} \geq g^{k+1}(m)$, so, unless (a) has already become true, we have $\max_i m_i^{t_k+D+1} \geq g^{k+2}(m)$, and $\max_i m_i^{t_{k+1}} \geq g^{k+2}(m)$ since $t_k + D + 1 \leq t_{k+1}$. Moreover, for all $t_{k+1} + D \leq t \leq t_{k+2}$, using Lemma 4.(a) and 4.(c) we have

$$\begin{aligned}
h\big(T_i^t \wedge \widetilde{G}^i\big) &\geq \min\Big\{ \min_i h\big(T_i^{t_{k+1}+D} \wedge \widetilde{G}^i\big) + t - t_{k+1} - D, \\
&\qquad \min_i m_i^{t_{k+1}+D} \Big\} \\
&\geq \min\Big\{ \min_i h\big(T_i^{t_{k+1}} \wedge \widetilde{G}^i\big) + t - t_{k+1} - D, \\
&\qquad \max_i m_i^{t_{k+1}} \Big\} \\
&\geq \min\big\{ t_{k+1} - (k+1)D + t - t_{k+1} - D, \\
&\qquad g^{k+2}(m) \big\} \\
&= t - (k+2)D. \blacksquare
\end{aligned}$$

Finally, we can prove our second main result, which is the finite-state counterpart of Theorem 1:

Theorem 7 Protocol (5) finite-state self-stabilizes the class of all networks to the predicate MB, with quiescence time at most $n_G + D_G + g^*(n_G + D_G)D_G$ on network G.

Proof. With the same notation as in Lemma 8, let $\bar{k} = \min\{k \mid g^k(m) \geq n + D\} \leq g^*(n + D)$. If $\bar{k} = 0$, then $m \geq n + D$, so using Lemmata 4.(a) and 5 we obtain the result. If $\bar{k} > 0$, note that $g^{\bar{k}-1}(m) < n + D$. and

$$t_{\bar{k}-1} + D \leq n + D + \bar{k}D \leq t_{\bar{k}}.$$

Thus, either we stabilized before time $t_{\bar{k}-1} \leq n + D + g^*(n + D)D$, or, by Lemma 8,

$$h\big(T_i^{n+D+\bar{k}D} \wedge \widetilde{G}^i\big) \geq n + D + \bar{k}D - \bar{k}D = n + D,$$

and since $D < n + D + \bar{k}D \leq t$, using Lemma 5 we obtain the thesis. $\blacksquare$

Note that, with respect to the infinite-state case, there is a loss given by the diameter times an arbitrarily slow function of $n + D$. This factor is due to the time necessary to detect locally that the guesses are too small, and to the time subsequently necessary to spread this knowledge to the whole network. Clearly, the resulting bound is $O(ng^*(n))$; thus, unless a precise space bound is required the loss in quiescence time can be reduced arbitrarily, and the gap with our lower bound (Corollary 2) can be made arbitrarily small. However, we conjecture that there is no universal finite-state protocol self-stabilizing to predicates with quiescence time $O(n)$.

6.1 Self-stabilizing clocks

The protocols of the previous sections allow self-stabilization to predicates. The second ingredient we need in order to self-stabilize to arbitrary behaviours is a synchronized clock. In the infinite-state case, the clock is provided at self-stabilization by $h\big(T_i^t\big)$, which is equal for all i and grows exactly by one at each step.

There are several results about self-stabilizing clock synchronization in the literature. For some of the more recent results, and a very good survey, see [14]. All known results, however, are not sufficiently general to be applied in our case (for instance, the only results about unidirectional networks are about rings).

We are now going to describe a finite-state self-stabilizing protocol that provides each processor with a clock modulo P, where $P \geq 1$ is a given constant. In our protocol, each processor i has two integer variables $M_i > 0$ and $0 \leq C_i < P(M_i)^2$, where C_i is the clock value (for the purposes of stabilization, however, we consider $C_i \bmod P$) and M_i is used as a guess to find a large enough multiple of P so to make the clocks stabilize quickly. Once again, we use a strictly increasing and inflationary update function $g : \mathbf{N} \to \mathbf{N}$; thus, the protocol is parameterized by the choice of g and P.

Denoting by C_i^t, M_i^t the values of clock and guess at time t, we define our protocol as follows[12]:

$$\boxed{\begin{aligned}
C_i^{t+1} &= \min_{j \looparrowright i} C_j^t + 1 \mod P\left(M_i^t\right)^2 \\
M_i^{t+1} &= \begin{cases} \max_{j \looparrowright i} M_j^t & \text{if } C_j^t \equiv_P C_i^t \text{ for all } j \to i \\ g\big(\max_{j \looparrowright i} M_j^t\big) & \text{otherwise.} \end{cases}
\end{aligned}} \tag{6}$$

The basic idea is that, until synchronization, we shall keep incrementing M_i so that, at some point, there will be "enough space" between the largest clock value and the smallest modulus. Note that as soon as all clocks have the same value *modulo* P, this property will remain true afterwards, because $(x \bmod kP) \bmod P = x \bmod P$. (Under this condition, an extra delay D will be necessary for all guesses to become equal, but this fact will not desynchronize the clocks.)

From now onwards, we let $M = \min_i M_i^0$ and we assume without loss of generality that $D > 0$. Note that after the first step all processors that are out-neighbours of those with minimum clock value will set their clock to no more than $\min_i C_i^0 + 1$, and that in D steps the upper bound will propagate to all processors (of course, clocks cannot increase more than 1 unit per step). More formally, since the clock value satisfies the hypotheses of Proposition 1, we have that for all $t \geq D$

$$C_i^t \leq \min_i C_i^0 + t \leq PM^2 + t.$$

We now prove the following

[12] For sake of simplicity, we defined our state space in such a way that it does not contain all pairs of guesses and clock values. If the reader is not at ease with this choice, he or she can equivalently substitute the first rule so to minimize $C_j^t \bmod P\big(M_j^t\big)^2$ instead of just C_j^t.

Lemma 9 For all $k \in \mathbf{N}$, one of the two following statements is true:

(i). for all i, j we have $C_i^{kD} \equiv_P C_j^{kD}$;

(ii). $\min_i M_i^{kD} \geq g^k(M)$.

Proof. For $k = 0$ we have $\min_i M_i^0 = M = g^0(M)$, and statement (ii) is true. Since statement (i) is stable, to make the inductive step we can assume that statement (ii) is true and statement (i) is false for k. But if there are two clocks with different values modulo P at time kD, then for every processor i there is a processor j satisfying $d(j,i) < D$ that will apply g at the next step (take a processor l with $C_l^t \not\equiv_P C_i^t$ minimizing $d(l,i) \leq D$, and let j be the first processor on a shortest path from l to i). Since for all processors j that apply g at time kD we have

$$M_j^{kD+1} \geq g\big(\min_i M_i^{kD}\big) \geq g^{k+1}(M),$$

in $D-1$ more steps (i.e., at time $kD+D$) this lower bound will propagate to all processors, that is, $\min_i M_i^{(k+1)D} \geq g^{k+1}(M)$. ∎

Theorem 8 Protocol (6) finite-state synchronizes (modulo P) all clocks within $g^*(D)D + D$ steps.

Proof. Let $k = g^*(D)$. At time kD either clocks are equal modulo P, or by Lemma 9 we have $M_i^{kD} \geq g^k(M)$ for all i. In this case,

$$\begin{aligned} P\big(M_i^{kD}\big)^2 - C_i^{kD} &\geq P\big(g^k(M)\big)^2 - (PM^2 + kD) \\ &= P\big(g^k(M) - M\big)\big(g^k(M) + M\big) - kD \\ &\geq Pg^k(0)\big(g^k(0) + 2M\big) - kD \\ &\geq PD(D+2) - g^*(D)D \\ &\geq D(D+2) - D^2 > D. \end{aligned}$$

Thus, since the guesses are nondecreasing, between time kD and $kD + D$ clock values will not "wrap", so by Proposition 1 at time $kD + D$ all clocks will have value $\min_i C_i^{kD} + D$. ∎

6.2 The finite-state characterization theorem

Theorem 9 Let $\mathscr{C}$ be a class of networks and S a behaviour on Y. Then, the following two conditions are equivalent:

- $\mathscr{C}$ can finite-state self-stabilize to S;
- for every fibration prime graph B there exists a periodic sequence $\boldsymbol{y}_B^0, \boldsymbol{y}_B^1, \ldots \in (Y^{n_B})^\omega$ such that for all $G \in \mathscr{C}$

$$\big(\boldsymbol{y}_{\hat{G}}^0\big)^{\mu_G}, \big(\boldsymbol{y}_{\hat{G}}^1\big)^{\mu_G}, \cdots \in S_G.$$

If the above holds, for every $g : \mathbf{N} \to \mathbf{N}$ strictly monotonic and inflationary, $\mathscr{C}$ can self-stabilize to S with quiescence time at most

$$n_G + \left[2 + g^*(n_G + D_G) + g^*(D_G)\right] D_G.$$

Proof. The condition is necessary by the Lifting lemma and by the fact that a finite-state behaviour is necessarily ultimately periodic. For sufficiency, we run in parallel the finite-state Protocol (5), which stabilizes every network to the predicate MB, and the finite-state clock synchronization protocol (6), using the period of the sequence associated to the current "candidate" minimum base (if $\mathscr{B}$ is undefined, the choice of the period is immaterial). As soon as the first protocol has reached stabilization, the period has the correct value, and, after the additional quiescence required for the clocks to synchronize, we self-stabilize to the required behaviour analogously to Theorem 5.

More precisely, at each step each processor applies the function $\mathscr{B}$ to obtain a guess B on the minimum base (and on the node j of the minimum base to which they would be mapped to); then, it sets the Y-part of its state space to the j-th component of $\boldsymbol{y}_B^h$, where h is the current clock value modulo the period of the sequence associated to B. The final statement is an immediate consequence of Theorems 7 and 8. ∎

In the case of predicates we have instead:

Theorem 10 Let $\mathscr{C}$ be a class of networks and S a predicate on Y. Then $\mathscr{C}$ can finite-state self-stabilize to S if and only if for every fibration prime graph B there exists a $\boldsymbol{y}_B \in Y^{n_B}$ such that $\big(\boldsymbol{y}_{\hat{G}}\big)^{\mu_G} \in S_G$ for all $G \in \mathscr{C}$. In such a case, $\mathscr{C}$ can self-stabilize to S with quiescence time at most $n_G + D_G + g^*(n_G + D_G)D_G$, where $g : \mathbf{N} \to \mathbf{N}$ is a strictly monotonic and inflationary function.

7 The case of unique identifiers

In the previous sections, we made no assumption on the nature of the node colouring; as an extreme case (the *uniform* case) all processors might have been indistinguishable (so their transition function would have always been the same). Since several papers about self-stabilization assume that processors have unique identifiers[13], in this section we want to focus our attention to this special case; a network has *unique identifiers* iff the node-colouring function is injective (this property is equivalent to saying that all processors are different, and use a different transition function).

As we shall see in a moment, we can fruitfully make use of the presence of unique identifiers to improve the efficiency of the algorithms and reduce the quiescence time. This fact evidences that the possibility of identifying each processor in a unique manner may reduce the time needed to self-stabilize a system; this improvement turns out to be particularly dramatic if the network has a small diameter. Moreover, Theorem 5 trivialises, as *every* behaviour admits a self-stabilizing protocol.[14]

Before presenting the modified algorithm, some comments are in order. If G has unique identifiers, then it is necessarily fibration prime; moreover, Theorem 3 may be rephrased in this case as follows:

[13] Typically, when studying topology reconstruction; note however that one can do self-stabilizing topology reconstruction (even if all processors are identical) in any class of fibration prime graphs (and only in such classes).

[14] Note that the asymmetry requirements of many papers on self-stabilization (e.g., [19, 3]) are such that all networks are fibration prime. This is the reason why they can prove self-stabilization to any desired behaviour.

Theorem 11 Let G and B be strongly connected graphs with unique identifiers such that $h\big(\widetilde{G}^i \wedge \widetilde{B}^j\big) \geq D_G + 1$. If no graph with less nodes than B satisfies this property, then $G \cong B$.

The proof is easy, as under the unique identifiers assumption the conditions on $\widetilde{G}^i$ and $\widetilde{B}^j$ imply $\widetilde{G}^i \cong \widetilde{B}^j$, so Theorem 3 can be applied. As a consequence, we can give a slightly different definition for the function $\mathscr{B}$: if T is a tree, $\mathscr{B}_u(T)$ is defined as the (unique, by Theorem 11) graph with minimum number of nodes in the set

$$\Big\{ G \;\Big|\; G \text{ has unique identifiers and } h(T) \geq D_G + 1 \text{ and } \exists i \in N_G \text{ such that } T \leq \widetilde{G}^i \Big\},$$

unless this set is empty (in which case $\mathscr{B}_u(T)$ is undefined). The function $\mathscr{B}_u$ has the following property, which is a special (stronger) version of Proposition 2:

Proposition 4 Let G be a graph with unique identifiers, and T be a tree. If $T \leq \widetilde{G}^i$ for some $i \in N_G$, then either $\mathscr{B}_u(T)$ is undefined, or $\mathscr{B}_u(T) \cong G$, and the latter happens iff $h(T) \geq D_G + 1$;

Proof. The proof follows from Theorem 11, as the existence of a smallest graph H such that $T \leq \widetilde{H}^j$ and $h(T) \geq D_H + 1$ would imply $G \cong H$, so $\mathscr{B}_u$ is undefined when $h(T) < D_G + 1$. The other implication is trivial. ∎

Our finite-state protocol is now as follows:

$$T_i^{t+1} = \begin{cases} T_i^t & \text{if } \mathscr{B}_u\big(T_i^t\big) \cong \mathscr{B}_u\big(U_i^t\big) \text{ and } T_i^t \leq U_i^t \\ U_i^t & \text{otherwise.} \end{cases} \tag{7}$$

To prove self-stabilization to MB we introduce the following definition:

$$c_i^t = \begin{cases} \infty & \text{if } \mathscr{B}_u(T_i^t) \cong G \text{ and } T_i^t \leq \widetilde{G}^i \\ h\big(T_i^t \wedge \widetilde{G}^i\big) & \text{otherwise,} \end{cases}$$

and we set $c^t = \min_i c_i^t$. This quantity measures the "minimum amount of correct levels" at time t, but assumes that sufficiently deep and correct trees have an infinite number of correct levels.

Lemma 10 For every $t \geq 0$ we have $c^{t+1} \geq c^t + 1$.

Proof. If $c^t = \infty$, then the only possible change of state for a processor is a truncation of its tree that leaves $\mathscr{B}_u$ untouched. Thus, $c^{t+1} = \infty$.

Assume now $c^t < \infty$, and note that if processor i sets T_i^{t+1} to T_i^t, then clearly $c_i^{t+1} = c_i^t \geq c^t$. On the other hand, if i sets T_i^{t+1} to U_i^t, then $c_i^{t+1} \geq c^t + 1$, as all in-neighbours of i (and i itself) have at least c^t correct levels.

We now show that all processors i such that $c_i^t = c^t$ fall in the latter case. Assume by contradiction that $\mathscr{B}_u\big(T_i^t\big) \cong \mathscr{B}_u\big(U_i^t\big)$ and $T_i^t \leq U_i^t$. Since $U_i^t \leq \widetilde{G}^i$ (as i has smallest correctness), $G \cong \mathscr{B}_u\big(U_i^t\big) \cong \mathscr{B}_u\big(T_i^t\big)$, so necessarily $T_i^t \not\leq \widetilde{G}^i$ (otherwise $c_i^t = \infty$). However, $T_i^t \leq \widetilde{G}^j$ for some $j \neq i$, so $T_i^t \wedge \widetilde{G}^i = \bot$. Thus, by (1)

$$-1 = h\big(T_i^t \wedge \widetilde{G}^i\big) = \min\big\{ h\big(U_i^t \wedge \widetilde{G}^i\big), h\big(T_i^t\big) \big\} \geq 0. \;∎$$

Theorem 12 Protocol (7) finite-state self-stabilizes the class of all networks with unique identifiers to the predicate MB, with quiescence time at most $D_G + 1$ on network G.

Proof. Assume by contradiction that for some $t > D$ we have $c^t < \infty$. Since the update rule always sets T_i^{t+1} to U_i^t when $h\big(U_i^t\big) < h\big(T_i^t\big)$, we can apply Proposition 1 using the tree height as function f, obtaining that $h\big(T_i^t\big) \leq c^0 + t$ for all i (as usual, after the first step trees of least correctness are truncated at height $c^0 + 1$). On the other hand, by Lemma 10 we have $c^0 + t \leq c^t \leq h\big(T_i^t\big)$. Thus, $h\big(T_i^t\big) = c^t \geq D + 1$ for all i, and, by Theorem 11, $c^t = \infty$, contradicting our assumption. State finiteness is immediate, as after stabilization $\max_i h\big(T_i^t\big)$ does not grow. ∎

Once again the above bound is tight: by suitably assigning unique identifiers to the family of graphs shown in Figure 8, it is easy to see that $\widetilde{G_{n,D}}^1$ and $\widetilde{H_{n,D}}^1$ are isomorphic up to level D, but not $D + 1$.

Note that in the infinite-state case, a synchronized clock C_i can be provided with the update rule $C_i^{t+1} = \min_{j \rightsquigarrow i} C_j^t + 1$; this clock will self-stabilize in D steps. Thus, by running the two protocols in parallel, we obtain:

Theorem 13 Let $\mathscr{C}$ be a class of networks with unique identifiers and S a behaviour on Y. Then $\mathscr{C}$ can self-stabilize to S with quiescence time at most $D_G + 1$ on network G.

On the other hand, in the finite-state case:

Theorem 14 Let $\mathscr{C}$ be a class of networks and S a behaviour on Y. Then $\mathscr{C}$ can finite-state self-stabilize to S if and only if S_G contains a periodic sequence for every $G \in \mathscr{C}$. In such a case, $\mathscr{C}$ can self-stabilize to S with quiescence time at most $3D_G + 1$ on network G.

Proof. We run in parallel Protocol (7) with the finite-state clock synchronization protocol (6). At each step t, let B be the current guessed minimum base (i.e., $\mathscr{B}_u(T_i^t)$); then we set P to the periodicity of the sequence associated to B and[15] $g(n) = n + D_B$. By Theorem 8 we obtain a synchronized clock in at most $2D_G$ steps after stabilization to MB (as $g^*(D_G) = g^*\big(D_B\big) = 1$ in this case). ∎

8 Examples

In this section we provide a number of examples, showing how, for classical and non-classical problems, we can easily characterize the classes in which self-stabilization is possible; we also give an informal specialization of our universal protocols, showing how to solve the problem "in practice" (of course, it may be

[15] Again, when B is undefined, the choice of D_B is immaterial.

possible to obtain by refinement more efficient versions of these protocols). We assume $Y = \{0, 1\}$, unless specified otherwise. Quiescence times can be easily derived from the results in the previous sections.

Leader election. In this problem, we must reach a state in which exactly one processor is a leader, and all other processors know they are not. We have that S_G is made of all tuples in Y^{n_G} containing exactly a 1. The necessary and sufficient condition on $\mathscr{C}$ for the existence of a self-stabilizing protocol is that for all fibration prime graphs B there is a node i such that for all $G \in \mathscr{C}$ satisfying $\hat{G} \cong B$ we have $\left|\mu_G^{-1}(i)\right| = 1$. Indeed, if the latter condition is satisfied then to the fibration prime graph B we can associate the tuple in Y^{n_B} containing exactly a 1 in position i, thus satisfying Theorem 10. On the other hand, if the condition is not true then any choice of $\boldsymbol{y}_B$ would produce a vector with zero or at least two 1's when lifted to at least one $G \in \mathscr{C}$. Note that in particular the condition is true for classes of fibration prime graphs or networks with unique identifiers, but these cases are very special. The protocol computes the minimum base B, and then the (necessarily unique) processor mapped to the node i of the condition above is elected.

Token teleportation (a.k.a. mutual exclusion). Processors have to *teleport* a token around, in such a way that only one single token exists at any time, and that on G each processor receives the token exactly each n_G steps. Thus, S_G contains the iterates of the sequence $\langle 1, 0, \ldots, 0\rangle$, $\langle 0, 1, \ldots, 0\rangle$, …, $\langle 0, 0, \ldots, 1\rangle$ and all the sequences obtained by index permutation. Note that this constraint must be satisfied even if the topology of G does not allow to intuitively "pass" the token along a link. The problem is solvable exactly on classes of fibration prime graphs: indeed, if a nonprime graph G exists in $\mathscr{C}$, then any sequence $\boldsymbol{y}_{\hat{G}}^t$ must have an item with a 1 in a position i such that $\left|\mu_G^{-1}(i)\right| > 1$, so Theorem 9 is not satisfied. On the other hand, we can just set $\boldsymbol{y}_B^t$ to the tuple in Y^{n_B} containing exactly a 1 in position $(t \bmod n_B) + 1$. The protocol computes the minimum base B (which is of course isomorphic to G), and establishes a synchronized clock (modulo $n_B = n_G$ in the finite-state case). At clock c processor $(c \bmod n_B) + 1$ holds the token, which is teleported to processor $(c+1 \bmod n_B)+1$ at the next step.

Complete topology reconstruction. Each processor must reconstruct the network it is running on, and select itself (in this case, Y is given by pairs of networks and natural numbers). More in detail, S_G contains tuples $\langle\langle H, i_1\rangle, \langle H, i_2\rangle, \ldots, \langle H, i_{n_G}\rangle\rangle$, where there is an isomorphism $\alpha : H \to G$ such that $\alpha(i_j) = j$. The problem is solvable exactly on classes of fibration prime graphs, as if a nonprime graph G exists in $\mathscr{C}$, then any choice for $\boldsymbol{y}_{\hat{G}}$ will give by lifting a tuple where a natural number appears two times as second component. On the other hand, on fibration prime graphs this predicate is a superset of MB, which is attainable.

Topology reconstruction. Each processor must reconstruct the network it is running on, *but it is not required that it selects itself.* It may seem strange, but the classes on which this problem is solvable are more than in the previous case. Indeed, the problem is solvable exactly on classes $\mathscr{C}$ such that for every fibration prime graph B there is at most a G in $\mathscr{C}$ such that $\hat{G} \cong B$ (the reader should be able to build a proof by this time). Every processor can clearly obtain the network from the minimum base.

Weak leader election. In this case, we require all processor to enter a third state $*$ (so $Y = \{0, 1, *\}$) if leader election is impossible on the network the protocol is running on. Thus, S_G is made of all tuples in Y^{n_G} containing either exactly a 1 if leader election is possible on $\{G\}$, or the only tuple $\langle *, *, \ldots, *\rangle$ otherwise. The necessary and sufficient condition on $\mathscr{C}$ for the existence of a self-stabilizing protocol is that whenever leader election is impossible in $\mathscr{C}_B = \{G \in \mathscr{C} \mid \hat{G} \cong B\}$ for a fibration prime graph B, then it is impossible in every $G \in \mathscr{C}_B$ (i.e., in $\{G\}$); the reader can easily check that this condition is much weaker than that needed in our first example (see, e.g., [8]). Indeed, if the condition holds we can set $\boldsymbol{y}_B = \langle *, *, \ldots, *\rangle$ when election is impossible in $\mathscr{C}_B$, or use the same $\boldsymbol{y}_B$ as in our first example otherwise (since election is possible in $\mathscr{C}_B$). On the other hand, if there is a fibration prime graph B such that election is impossible in $\mathscr{C}_B$ but possible on some $G \in \mathscr{C}_B$, any assignment to $\boldsymbol{y}_B$ would not suit S_G. The resulting protocol computes the minimum base B; then, either election is impossible in $\mathscr{C}_B$, and all processors "surrender", or we proceed as in our first example.

Lower bounded synchronized clock. All processors are required to hold a synchronized clock, increasing of one unit at each step, whose value is greater than n_G^2; more precisely, S_G is made by the sequence $\langle n_G^2, n_G^2, \ldots, n_G^2\rangle$, $\langle n_G^2 + 1, n_G^2 + 1, \ldots, n_G^2 + 1\rangle$, … and its suffixes. Clearly any class can self-stabilize to this behaviour, but the interesting point is that the quiescence time is in general greater than $n + D$ (i.e., the maximum in Theorem 5 is realized by T_G). This fact is true of every protocol, and in particular of ours, as the following argument shows: consider an arbitrary protocol self-stabilizing to S on the class of unidirectional uniform rings. On the ring with one processor, starting from a state x we obtain a sequence of states x^0, x^1, … such that for a certain T we have $\pi\big(x^T\big)$, $\pi\big(x^{T+1}\big)$, … is in S. Thus, self-stabilization on the ring of $n \geq \pi\big(x^T\big)$ processors is impossible before $n^2 - \pi\big(x^T\big) + T \sim n^2$ steps, as one can easily see starting a computation with local initial state equal to x for all processors. Note that by choosing a function faster than squaring we can enlarge the gap with $n + D$ arbitrarily.

9 Conclusions

This paper presented an organic theory of self-stabilization on synchronous networks. In Table 3 we summarize the bounds we obtained for our universal protocols. As far as the minimum base construction is concerned, our bounds are all tight except for the general finite-state case, were we have an asymptotically nearly optimal bound (the upper bound is $O(ng^*(n))$ with g^* arbitrarily slow, and the lower bound is $\Omega(n)$). It is a relevant open problem to prove or disprove the conjecture that there is no finite-state universal protocol with quiescence time $O(n)$, but the fact that there are protocols arbitrarily close to that bound suggests that the task is not easy.

	Minimum base construction	Clock synchronization phase
Inf. states	$n + D$ (tight)	
Inf. states, unique id's	$D + 1$ (tight)	
Fin. states	$n + D + g^*(n + D)D$	$D + g^*(D)D$
Fin. states, unique id's	$D + 1$ (tight)	$2D$

Table 3: A summary of the bounds on quiescence time.

A natural question arises about the possibility of emulating the protocols we described in an asynchronous message-passing system. There is a standard equivalence result between synchronous shared-memory and asynchronous message-passing anonymous networks proved by Yamashita and Kameda [22] that can be easily extended to the level of generality adopted in this paper, but the extension to self-stabilizing systems would require the design of self-stabilizing simulations of shared memory (such as those described in [1, 16]) that do not require any additional information (e.g., distinguished incoming links). To our knowledge, no such protocols exist at this time.

The synchronous scheduler assumed in this paper is less powerful than the *interleaved* (a.k.a. central daemon) scheduler often assumed in the literature; the latter guarantees that exactly one processor is activated at each step, and this property can be used to introduce asymmetry (e.g., rings of prime size have self-stabilizing leader election algorithms using interleaved activation, but not using synchronous activation). However, asynchronous message-passing and synchronous shared-memory systems cannot in general emulate this powerful feature, as shown in the case of anonymous networks (and thus *a fortiori* for self-stabilizing systems) in [8]. Of course, this difference is immaterial once unique identifiers are assumed, or created using randomness (as in [4, 17, 15]). Note that the distribution of such unique identifiers would allow one to use the faster protocols of Section 7; however, the existing algorithms use some knowledge (e.g., bidirectionality) that we do not assume, and thus they can be applied only if the class under consideration satisfies the additional necessary requirements.

In [11] we proved that the minimum base construction protocols work also with a completely asynchronous scheduler (but we could not prove optimality of their quiescence time). Thus, a theory similar to the one outlined here can be pursued for networks with an asynchronous or interleaved scheduler, using the fibration-theoretic notions described in [8, 10]. In each case, the computability results for classes of anonymous networks with bounded size can always be turned immediately into results about self-stabilization by substituting the computation of the minimum base (that uses the knowledge of the size bound and assumes correct views) with the guess given by the function $\mathscr{B}$; thus, characterizations for specific problems (such as those given in the abovementioned papers) can be easily turned into characterization for self-stabilization.

As a final note, we remark that particular conditions on the colouring of the arcs can make more behaviours attainable. For instance, if, as it happens in a very common model, all links are bidirectional and locally distinguishable by the processors, then the simple presence of a leader makes a network fibration prime (see [9]). Thus, in this case the presence of a single distinguished processor makes it possible to self-stabilize to *any* behaviour. However, this is no longer true if processors cannot distinguish their outgoing links, as it is immediate to check using the tools described in this paper.

References

[1] Yehuda Afek and Geoffrey M. Brown. Self-stabilization over unreliable communication media. *Distributed Computing*, 7(1):27–34, 1993.

[2] Yehuda Afek and Shlomi Dolev. Local stabilizer. In *ISTCS: 5th Israeli Symposium on the Theory of Computing and Systems*, pages 74–84. IEEE Press, 1997.

[3] Yehuda Afek, Shay Kutten, and Moti Yung. Memory-efficient self-stabilizing protocols for general networks. In *Proc. of the International Workshop on Distributed Algorithms*, number 486 in Lecture Notes in Computer Science, pages 15–28. Springer–Verlag, 1991.

[4] Yehuda Afek, Shay Kutten, and Moti Yung. The local detection paradigm and its application to self-stabilization. *Theoret. Comput. Sci.*, 186(1–2):199–229, 1997.

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

[6] Baruch Awerbuch, Boaz Patt–Shamir, and George Varghese. Self-stabilization by local checking and correction. In *Proc. 32nd Symposium on Foundations of Computer Science*, pages 268–277. IEEE Press, 1991.

[7] Baruch Awerbuch and George Varghese. Distributed program checking: A paradigm for building self-stabilizing distributed protocols. In *Proc. 32nd Symposium on Foundations of Computer Science*, pages 258–267. IEEE Press, 1991.

[8] 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.

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

[10] 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.

[11] 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.

[12] James E. Burns, Mohamed G. Gouda, and Raymond E. Miller. Stabilization and pseudo-stabilization. *Distr. Comput.*, 7:35–42, 1993.

[13] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *CACM*, 17(11):643–644, 1974.

[14] Shlomi Dolev. Possible and impossible self-stabilizing digital clock synchronization in general graphs. *Journal of Real–Time Systems*, 12(1):95–107, 1997.

[15] Shlomi Dolev. Optimal time self-stabilization in uniform dynamic systems. *Parallel Process. Lett.*, 8:7–18, 1998.

[16] Shlomi Dolev, Amos Israeli, and Shlomo Moran. Resource bounds for self-stabilizing message-driven protocols. *SIAM J. Comput.*, 26(1):273–290, 1997.

[17] Shlomi Dolev, Amos Israeli, and Shlomo Moran. Uniform dynamic self-stabilizing leader election. *IEEE Transactions on Parallel and Distributed Systems*, 8(4):424–440, 1997.

[18] 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.

[19] Shmuel Katz and Kenneth J. Perry. Self-stabilizing extensions for message-passing systems. *Distr. Comput.*, 7:17–26, 1993.

[20] Nancy Norris. Universal covers of graphs: Isomorphism to depth $n - 1$ implies isomorphism to all depths. *Discrete Appl. Math.*, 56:61–74, 1995.

[21] Masafumi Yamashita and Tiko Kameda. Computing functions on asynchronous anonymous networks. *Math. Systems Theory*, 29(4):331–356, 1996. See also [23].

[22] 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.

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