Stably computable properties of network graphs

Dana Angluin[*] James Aspnes[*†] Melody Chan[*] Michael J. Fischer[*‡]
Hong Jiang[*§] René Peralta[*‡]

January 17, 2005

Abstract

We consider a scenario in which anonymous, finite-state sensing devices are deployed in an ad-hoc communication network of arbitrary size and unknown topology, and explore what properties of the network graph can be stably computed by the devices. We show that they can detect whether the network has degree bounded by a constant d, and, if so, organize a computation that achieves asymptotically optimal linear memory use. We define a model of stabilizing inputs to such devices and show that a large class of predicates of the multiset of final input values are stably computable in any weakly-connected network. We also show that nondeterminism in the transition function does not increase the class of stably computable predicates.

1 Introduction

In some applications, a large number of sensors will be deployed without fine control of their locations and communication patterns in the target environment. To enable the distributed gathering and processing of information, the sensors must constitute themselves into an ad-hoc network and use it effectively. A fundamental question in this context is whether there are protocols that determine enough about the topological properties of this network to exploit its full potential for distributed computation, when sensors are severely limited in their computational power.

We consider a model introduced in [2] in which communication is represented by pairwise interactions of anonymous finite-state devices in networks of finite but unbounded size. These systems correspond to sensor networks with intermittent two-way communications between nearby nodes. A **communication graph** describes which pairs of nodes are close enough to eventually interact. Our goal is to explore what graph-theoretic properties of the communication graph are **stably computable**, where a property is stably computable if all sensors eventually converge to

[*]Yale University Department of Computer Science.
[†]Supported in part by NSF grants CCR-9820888, CCR-0098078, CNS-0305258, and CNS-0435201.
[‡]Supported in part by NSF grant CSE-0081823.
[§]Supported by NSF grant ITR-0331548

the correct answer. The model assumes that the devices have no identifiers and only a few bits of memory each. Because each device is so limited, their collective ability to achieve nontrivial computation must be based on their capacity to organize a distributed computation in the network.

In this setting, the structure of the network has a profound influence on its computational potential. If n is the number of vertices in the communication graph, previous results show that $O(\log n)$ bits of memory are sufficient for a nondeterministic Turing machine to simulate any protocol in the all-pairs communication graph, in which every pair of vertices is joined by an edge [2]. In contrast, we give a protocol that can determine whether the communication graph is a directed cycle and if so, use it as a linear memory of $O(n)$ bits, which is asymptotically optimal in terms of memory capacity. More generally, we show that for every d there is a protocol that can organize any communication graph of maximum degree d into a linear memory of $O(n)$ bits, also asymptotically optimal.

For general communication graphs, we show that any property that is determined by the existence of a fixed finite subgraph is stably computable, as are Boolean combinations of such properties. In addition, there are protocols to compute the following graph properties: whether the communication graph G is a directed star, whether G is a directed arborescence, whether G contains a directed cycle, and whether G contains a directed cycle of odd length. Furthermore, for any positive integer d, there is a protocol that stabilizes to a proper d-coloring of any d-colorable graph, but does not stabilize if the graph is not d-colorable.

In the model of [2], the sensor readings were all assumed to be available as inputs at the start of the computation. In this paper, we extend the model to allow for a more realistic scenario of stabilizing inputs, that may change finitely many times before attaining a final value. In addition to allowing fluctuations in the inputs, these results allow composition of stabilizing protocols: a protocol that works with stabilizing inputs can use the stabilizing outputs of another protocol. We generalize two fundamental theorems to the case of stabilizing inputs: all the **Presburger-definable** predicates are stably computable in the all-pairs graph, and any predicate stably computable in the all-pairs graph is stably computable in any weakly connected graph with the same number of nodes.

Another powerful tool for the design of protocols is to permit a nondeterministic transition function; we give a simulation to show that this does not increase the class of stably computable predicates.

1.1 Other related work

Population protocols and similar models as defined in [1, 2, 7] have connections to a wide range of theoretical models and problems involving automata of various kinds, including Petri nets [3, 8, 9, 19], semilinear sets and Presburger expressions [11], vector addition systems [13], the Chemical Abstract Machine [4, 14] and other models of communicating automata [5, 20]. See [2] for a more complete discussion of these connections.

The potential for distributed computation during aggregation of sensor data is studied in [15, 18], and distributed computation strategies for conserving resources in tracking targets in [10, 22]. Issues of random mobility in a wireless packet network are considered in [12].

Passively mobile sensor networks have been studied in several practical application contexts. The **smart dust** project [6, 17] designed a cloud of tiny wireless microelectromechanical sensors (MEMS) with wireless communication capacity, where each sensor or "mote" contains sensing units, computing circuits, bidirectional wireless capacity, and a power supply, while being inexpensive enough to deploy massively. More recently, Thorstensen *et al.* [21] introduced a low-cost, wireless communication network system called the **Electronic Shepherd**, used to monitor the state of a flock of sheep using short-hop communication between sensors augmented by the flocking behavior of the sheep. This work, focused on actual deployment rather than theoretical results, demonstrates the practicality of a passively-mobile communications model.

2 The model of stable computation

We represent a network communication graph by a directed graph $G = (V, E)$ with n vertices numbered 1 through n and no multi-edges or self-loops. Each vertex represents a finite-state sensing device, and an edge (u, v) indicates the possibility of a communication between u and v in which u is the **i**nitiator and v is the **r**esponder.[1] We assume G is *weakly connected*, that is, between any pair of nodes there is a path (disregarding the direction of the edges in the path). The **all-pairs graph** contains all edges (u, v) such that $u \neq v$.

We first define protocols without inputs, which is sufficient for our initial results on graph properties, and extend the definition to allow stabilizing inputs in Section 4.

A **protocol** consists of a finite set of **states** Q, an **initial state** $q_0 \in Q$, an **output function** $O : Q \to Y$, where Y is a finite set of output symbols, and a **transition function** δ that maps every pair of states (p, q) to a nonempty set of pairs of states. If $(p', q') \in \delta(p, q)$, we call $(p, q) \mapsto (p', q')$ a **transition**.

The transition function, and the protocol, is **deterministic** if $\delta(p, q)$ always contains just one pair of states. In this case we write $\delta(p, q) = (p', q')$ and define $\delta_1(p, q) = p'$ and $\delta_2(p, q) = q'$.

A **configuration** is a mapping $C : V \to Q$ specifying the state of each device in the network. We assume that there is a global start signal transmitted simultaneously to all the devices, e.g., from a base station, that puts them all in the initial state and starts the computation. The **initial configuration** assigns the initial state q_0 to every device.

Let C and C' be configurations, and let u, v be distinct nodes. We say that C goes to C' via **pair** $e = (u, v)$, denoted $C \xrightarrow{e} C'$, if the pair $(C'(u), C'(v))$ is in $\delta(C(u), C(v))$ and for all $w \in V - \{u, v\}$ we have $C'(w) = C(w)$. We say that C can go to C' in one step, denoted $C \to C'$, if $C \xrightarrow{e} C'$ for some edge $e \in E$. We write $C \xrightarrow{*} C'$ if there is a sequence of configurations $C = C_0, C_1, \ldots, C_k = C'$, such that $C_i \to C_{i+1}$ for all i, $0 \leq i < k$, in which case we say that C' is **reachable** from C.

A **computation** is a finite or infinite sequence of population configurations $C_0, C_1, \ldots$ such that for each i, $C_i \to C_{i+1}$. An infinite computation is **fair** if for every pair of population configurations C and C' such that $C \to C'$, if C occurs infinitely often in the computation, then C' also occurs

[1] The distinct roles of the two devices in an interaction is a fundamental assumption of asymmetry in our model; symmetry-breaking therefore does not arise as a problem within the model.

infinitely often in the computation. We remark that by induction, if $C \xrightarrow{*} C'$ and C occurs infinitely often in a fair computation, then C' also occurs infinitely often in a fair computation.

2.1 Leader election

As an example, we define a simple deterministic leader election protocol that succeeds in any network communication graph. The states of the protocol are $\{1, 0\}$ where 1 is the initial state. The transitions are defined by

$$\begin{array}{lccc} (1) & (1,1) & \mapsto & (0,1) \\ (2) & (1,0) & \mapsto & (0,1) \\ (3) & (0,1) & \mapsto & (1,0) \\ (4) & (0,0) & \mapsto & (0,0) \end{array}$$

We think of 1 as the leader mark. In every infinite fair computation of this protocol starting with the initial configuration in any communication graph, after some finite initial segment of the computation, every configuration has just one vertex labeled 1 (the leader), and every vertex has label 1 in infinitely many different configurations of the computation. Thus, eventually there is one "leader" mark that hops incessantly around the graph, somewhat like an ancient English king visiting the castles of his lords. Note that in general the devices have no way of knowing whether a configuration with just one leader mark has been reached yet.

2.2 The output of a computation

Our protocols are not designed to halt, so there is no obvious fixed time at which to view the output of the computation. Rather, we say that the output of the computation stabilizes if it reaches a point after which no device can subsequently change its output value, no matter how the computation proceeds thereafter. Stability is a global property of the graph configuration, so individual devices in general do not know when stability has been reached.[2]

We define an **output assignment** y as a mapping from V to the output symbols Y. We extend the output map O to take a configuration C and produce an output assignment $O(C)$ defined by $O(C)(v) = O(C(v))$. A configuration C is said to be **output-stable** if $O(C') = O(C)$ for all C' reachable from C. Note that we do not require that $C = C'$, only that their output assignments be equal. An infinite computation **output-stabilizes** if it contains an output-stable configuration C, in which case we say that it **stabilizes to output assignment** $y = O(C)$. Clearly an infinite computation stabilizes to at most one output assignment.

The output of a finite computation is the output assignment of its last configuration. The output of an infinite computation that stabilizes to output assignment y is y; the output is undefined if the computation does not stabilize to any output assignment. Because of the nondeterminism inherent in the choice of encounters, the same initial configuration may lead to different computations that stabilize to different output assignments.

[2] With suitable stochastic assumptions on the rate at which interactions occur, it is possible to bound the expected number of interactions until the output stabilizes, a direction explored in [2].

2.3 Graph properties

We are interested in what properties of the network communication graph can be stably computed by protocols in this model. A **graph property** is a function P from graphs G to the set $\{0, 1\}$ where $P(G) = 1$ if and only if G has the corresponding property. We are interested in families of graphs, $\mathcal{G}_1, \mathcal{G}_2, \ldots$, where $\mathcal{G}_n$ is a set of network graphs with n vertices. The **unrestricted** family of graphs contains all possible network communication graphs. The **all-pairs** family of graphs contains for each n just the all-pairs graph with n vertices. For every d, the family of d **degree-bounded** graphs contains all the network communication graphs in which the in-degree and out-degree of each vertex is bounded by d. Similarly, the family of d **colorable** graphs contains all the network communication graphs properly colorable with at most d colors.

We say that a protocol $\mathcal{A}$ stably computes the graph property P in the family of graphs $\mathcal{G}_1, \mathcal{G}_2, \ldots$ if for every graph G in the family, every infinite fair computation of $\mathcal{A}$ in G starting with the initial configuration stabilizes to the constant output assignment equal to $P(G)$. Thus the output of every device stabilizes to the correct value of $P(G)$.

3 Example: is G a directed cycle?

In this section, we show that the property of G being a directed cycle is stably computable in the unrestricted family of graphs. Once a directed cycle is recognized, it can be organized (using leader-election techniques) to simulate a Turing machine tape of n cells for the processing of inputs, which vastly increases the computational power over the original finite-state devices, and is optimal with respect to the memory capacity of the network.

Theorem 1 *Whether G is a directed cycle is stably computable in the unrestricted family of graphs.*

Proof sketch: A weakly connected directed graph G is a directed cycle if and only if the in-degree and out-degree of each vertex is 1.

Lemma 2 *Whether G has a vertex of in-degree greater than 1 is stably computable in the unrestricted family of graphs.*

We give a protocol to determine whether some vertex in G has in-degree at least 2. The protocol is deterministic and has 4 states: $\{-, I, R, Y\}$, where $-$ is the initial state, I and R stand for "initiator" and "responder" and Y indicates that there is a vertex of in-degree at least 2 in the graph. The transitions as follows, where x is any state and unspecified transitions do not change their inputs:

$$\begin{array}{lccc} (1) & (-,-) & \mapsto & (I,R) \\ (2) & (I,R) & \mapsto & (-,-) \\ (3) & (-,R) & \mapsto & (Y,Y) \\ (4) & (Y,x) & \mapsto & (Y,Y) \\ (5) & (x,Y) & \mapsto & (Y,Y) \end{array}$$

The output map takes Y to 1 and the other states to 0. State Y is contagious, spreading to all states if it ever occurs. If every vertex has in-degree at most 1 then only transitions of types (1) and (2) can occur, and if some vertex has in-degree at least 2, then in any fair computation some transition of type (3) must eventually occur.

An analogous four-state protocol detects whether any vertex has out-degree at least 2. We must also guarantee that every vertex has positive in-degree and out-degree.

Lemma 3 *Whether G has a vertex of in-degree 0 is stably computable in the unrestricted family of graphs.*

There is a deterministic eight-state protocol for this predicate, which we omit for space reasons. The following deterministic two-state protocol stably labels each vertex with Z if it has in-degree 0 and P if it has in-degree greater than 0. The initial state is Z. The transitions are given by the following, where x and y are any states:

$$(1) \quad (x, y) \quad \mapsto \quad (x, P)$$

To detect whether all states are labeled P in the limit, we would like to treat the outputs of this protocol as the inputs to a another protocol to detect any occurrences of Z in the limit. However, to do this, the protocol to detect any occurrences of Z must cope with inputs that may change before they stabilize to their final values. In the next section we show that all the Presburger predicates (of which the problem of Z detection is a simple instance) are stably computable with such "stabilizing inputs" in the unrestricted family of graphs, establishing the existence of the required protocol.

Similarly, there is a protocol that stably computes whether every vertex has out-degree at least 1. By running all four protocols in parallel, we may stably compute the property: does every vertex have in-degree and out-degree exactly 1? Thus, there is a protocol that stably computes the property of G being a directed cycle for the unrestricted family of graphs, proving Theorem 1. ∎

These techniques generalize easily to recognize other properties characterized by conditions on vertices having in-degrees or out-degrees of zero or one. A directed line has one vertex of in-degree zero, one vertex of out-degree zero, and all other vertices have in-degree and out-degree one. An out-directed star has one vertex of in-degree zero and all other vertices of in-degree one and out-degree zero, and similarly for an in-directed star. An out-directed arborescence has one vertex of in-degree zero and all other vertices have in-degree one, and similarly for an in-directed arborescence.

Theorem 4 *The graph properties of being a directed line, a directed star, or a directed arborescence are stably computable in the unrestricted family of graphs.*

In a later section, we generalize Theorem 1 to show that for any d there is a protocol to recognize whether the network communication graph has in-degree and out-degree bounded by d and to organize it as $O(n)$ cells of linearly ordered memory if so. We now turn to the issue of inputs.

4 Computing with stabilizing inputs

We define a model of stabilizing inputs to a network protocol, in which the input to each node may change finitely many times before it stabilizes to a final value. We are interested in what predicates of the final input assignment are stably computable. An important open question is whether any predicate stably computable in a family of graphs is stably computable with stabilizing inputs in the same family of graphs.

Though we do not fully answer this question, we show that a large class of protocols can be adapted to stabilizing inputs by generalizing two theorems from [2] to the case of stabilizing inputs: all Presburger predicates are stably computable with stabilizing inputs in the all-pairs family of graphs, and any predicate that can be stably computed with stabilizing inputs in the all-pairs family of graphs can be stably computed with stabilizing inputs in the unrestricted family of graphs.

We assume that there is a finite set of input symbols, and each device has a separate input port at which its current input symbol (representing a sensed value) is available at every computation step. Between any two computation steps, the inputs to any subset of the devices may change arbitrarily.

Formally, we extend the definition of protocols by adding finite set X of input symbols, and stipulating that the transition function will map $(Q \times X) \times (Q \times X)$ to a singleton subset of $Q \times Q$ (for deterministic protocols), or a nonempty subset of $Q \times Q$ (for nondeterministic protocols.) A **configuration** C is a mapping of V to $(Q \times X)$, specifying the state and input for every vertex in the graph. Using the standard projection functions, $\pi_1(C(u))$ is the state of u in C and $\pi_2(C(u))$ is the input of u in C. An **initial configuration** has every vertex in the initial state, and arbitrary inputs. The **output** of a configuration is obtained by applying the output map to the state components of the configuration.

We revise the definition of a one-step transition between configurations as follows. Let C and C' be configurations, and let u, v be distinct nodes. We say that C goes to C' via **pair** $e = (u, v)$, denoted $C \xrightarrow{e} C'$, if $(\pi_1(C'(u)), \pi_1(C'(v))) \in \delta(C(u), C(v))$ and for all $w \in V - \{u, v\}$, we have $\pi_1(C'(w)) = \pi_1(C(w))$. That is, the states of u and v in C' have been updated according to δ, using the states and inputs of u and v in C, and the state of every other vertex w is unchanged from C to C'. Note that there is no constraint on the inputs in C'.

Computations are defined as previously. However, we need to distinguish those computations in which the inputs stabilize. We say a computation $C_0, C_1, C_2, \ldots$ has **stabilizing inputs** if there is some finite step k after which the input to each vertex does not change. Thus, there is a **final input assignment** $x : V \to X$ such that $x(v) = \pi_2(C_l(v))$ for every $v \in V$ and every $l \geq k$. A computation $C_0, C_1, C_2, \ldots$ has **unchanging inputs** if for each v, the input to v never changes, that is, $\pi_2(C_k(v)) = \pi_2(C_0(v))$ for all k. Unchanging inputs are a special case of stabilizing inputs.

A computation $C_0, C_1, C_2, \ldots$ with stabilizing inputs is **fair** if for every configuration C that occurs infinitely often in the computation, and every configuration C' such that the input assignment in C' is equal to the input assignment in C and $C \to C'$, C' also occurs infinitely often in the computation. These definitions specialize to the previous ones (with no input) when X is a single letter alphabet.

We now consider predicates defined not just on graphs, but on graphs with vertices labeled by input symbols. A **labeled graph property** P is a mapping from pairs (G, x) to $\{0, 1\}$, where G

is a network graph and x is an input assignment for G. A protocol $\mathcal{A}$ stably computes the labeled graph property P with stabilizing inputs in the family of graphs $\mathcal{G}_1, \mathcal{G}_2, \ldots$ if for every n and every graph $G \in \mathcal{G}_n$, and every input assignment x to G, every fair computation of $\mathcal{A}$ in G that starts from an initial configuration and has inputs stabilizing to final assignment x stabilizes to the output $P(G, x)$. We will use the term **predicate** to mean a labeled graph property hereafter. We note that in the special case of the all-pairs graph, the graph structure is completely symmetric and the stably computable predicates depend only on the multiset of inputs determined by x.

By running several protocols in parallel, and defining the output at each vertex to be a Boolean combination of the outputs of the individual protocols, we have the following.

Lemma 5 *Any Boolean combination of a finite set of predicates stably computable with stabilizing inputs in a family of graphs $\mathcal{G}_1, \mathcal{G}_2, \ldots$ is stably computable with stabilizing inputs in the same family of graphs.*

4.1 Nondeterministic protocols

To simplify the proofs of the existence of protocols, we first show how nondeterministic protocols can be transformed into deterministic protocols. Intuitively, this is because a deterministic protocol can exploit the nondeterminism inherent in the choice of which interaction occurs.

A nondeterministic d-coloring protocol. To illustrate the power of nondeterministic protocols, we describe a very simple protocol that is guaranteed to stabilize on a d-coloring of any network communication graph in the d-colorable family of graphs. The states Q are the integers 0 through $d-1$ inclusive, thought of as colors, with initial state 0. The goal is to stabilize in a configuration in which no two adjacent vertices have the same state. For each $i \in Q$ there is one transition rule:

$$(1) \quad (i, i) \quad \mapsto \quad (j, k)$$

where j and k are any elements of Q. All other transitions are the identity. Transitions of type (1) select a "conflict edge," that is, an edge joining two vertices currently assigned the same color, and arbitrarily recolor the two endpoints. The fairness condition guarantees that the appropriate (j, k)'s will eventually be chosen.

To see that if G is d-colorable, any fair computation must stabilize to a correct d-coloring of G, observe that from any configuration containing a conflict edge there is a reachable configuration containing no conflict edges. If G is not d-colorable, the coloring never stabilizes in a fair computation.

Theorem 6 *For every nondeterministic protocol $\mathcal{A}$ there exists a deterministic protocol $\mathcal{B}$ such that if $\mathcal{A}$ stably computes a predicate P with stabilizing inputs in a family of graphs, then $\mathcal{B}$ also stably computes P with stabilizing inputs in the same family of graphs.*

Proof sketch: Let $\mathcal{A}$ be a nondeterministic protocol with states Q, initial state q_0, input symbols X, output symbols Y, output map O, and transition function δ. We describe a simulation of $\mathcal{A}$ that works in graphs with at least 3 vertices. For smaller graphs, we combine the simulation

with a protocol that tests whether $n = 1$ or $n = 2$ (using the fact that Presburger predicates of n can be stably computed) and stabilizes to the correct results in those cases.

Let m be the maximum cardinality of any of the sets $\delta((q,a),(q',a'))$ for $(q,a),(q',a') \in Q \times X$. For each pair of labels $(q,a),(q',a') \in Q \times X$, select an arbitrary surjective function $f_{(q,a),(q',a')}$ mapping $\{0,1,\ldots,m-1\}$ to $\delta((q,a),(q',a'))$.

We describe a protocol $\mathcal{B}$ to simulate $\mathcal{A}$. The states consist of three components: (1) a leader mark $*$ or its absence $-$. (2) a state $q \in Q$. (3) a choice counter, consisting of an integer between 0 and $m-1$ inclusive. The output map O' is $O'(xqc) = O(q)$. The initial state of every device is $*q_0 0$.

The transitions are

$$\begin{array}{lrcl}
(1) & ((*qc,a),(*q'c',a')) & \mapsto & (-qc,*q'c') \\
(2) & ((*qc,a),(-q'c',a')) & \mapsto & (-qc,*q'(c'+1)) \\
(3) & ((-q'c',a'),(*qc,a)) & \mapsto & (*q'(c'+1),-qc) \\
(4) & ((-qc,a),(-q'c',a')) & \mapsto & (-rc,-r'c')
\end{array}$$

where the increments are made modulo m and the pair of states (r,r') is the element of $\delta((q,a),(q',a'))$ selected by the function $f_{(q,a),(q',a')}(c)$. Thus, in transitions of type (4), the value of the choice counter of the initiator is used to make a deterministic choice of an element of $\delta((q,a),(q',a'))$. The role of the leader mark, or nondeterminizer, is to hop around the graph incrementing choice counters as it goes; this is the function of the first three types of transitions. ∎

4.2 The Presburger predicates

The Presburger graph predicates form a useful class of predicates on the multiset of input symbols, including such things as "all the inputs are a's", "at least 5 inputs are a's", "the number of a's is congruent to 3 modulo 5", and "twice the number of a's exceeds three times the number of b's" and Boolean combinations of such predicates. Every Presburger graph predicate is stably computable with unchanging inputs in the all-pairs family of graphs [2]. The main result of this subsection is the following generalization.

Theorem 7 *Every Presburger graph predicate is stably computable with stabilizing inputs in the family of all-pairs graphs.*

We define the Presburger graph predicates as those expressible in the following expression language.[3] For each input symbol $a \in X$, there is a variable $\#(a)$ that represents the number of occurrences of a in the input. A **term** is a linear combination of variables with integer coefficients, possibly modulo an integer. An **atom** is two terms joined by one of the comparison operators: $<, \leq, =, \geq, >$. An **expression** is a Boolean combination of atoms.

Thus, if the input alphabet $X = \{a,b,c\}$, the predicate "all the inputs are a's" can be expressed as $\#(b) + \#(c) = 0$, the predicate that the number of a's is congruent to 3 modulo 5 can be expressed as $(\#(a) \bmod 5) = 3$, and the predicate that twice the number of a's exceeds three times the number of b's by $2\#(a) > 3\#(b)$.

[3] These are closely related to the Presburger integer predicates defined in [2]. Details will be given in the full paper.

Majority To indicate the ideas of the general proof, we show how to generalize the basic predicate of majority to stabilizing inputs. The input and output alphabets are $\{0, 1\}$. The value of the majority predicate is 1 if there are more 1's than 0's in the final input assignment; otherwise, it is 0. We describe a deterministic protocol with states consisting of a leader bit, a most recent input bit, and a counter with integer values from -2 to 2. The initial state has leader bit equal to 1, most recent input bit equal to 0, and counter equal to -1. The output is 1 if the counter is positive; otherwise, it is 0.

The following invariants are preserved in the computation: (1) any vertex with leader bit 1 and a negative counter has most recent input bit 0 and any vertex with leader bit 1 and a positive counter has most recent input bit 1, and (2) the sum of the counters of all vertices with leader bit equal to 1 is equal to the number of 1's in the most recent input bits minus the number of 0's in the most recent input bits. This will show that the following protocol correctly computes majority.

In an interaction, the two devices first update their most recent input bit as follows. If the current input is equal to the most recent input bit, no updating is necessary. Otherwise, if the leader bit is 0, the leader bit is set to 1 and the counter is set to 2 if the current input is 1 and -2 if the current input is 0. If the leader bit is 1, then 2 is added to the counter if the current input is 1, or 2 is subtracted from the counter if the current input is 0. (We must show that this does not cause the counter to overflow.) In either case, the most recent input bit is then set to the current input.

After both agents have performed this update, they interact as follows. If both have leader bits equal to 1, if their counter values are both positive or both negative, no further changes are made. If they have opposite signs, or one or both are 0, then both their counters are set to the sum of their counters. One of the vertices sets its leader bit to 0, subject to the condition that if the remaining leader's counter is negative, then its most recent input bit is 0, and if it is positive, its most recent input bit is 1. (We must show that this condition can be satisfied.) If only one has leader bit equal to 1, then the other copies its counter value. If neither has leader bit equal to 1, then there is no change of state.

If we consider any fair computation of this protocol from an initial configuration with stabilizing inputs, after some finite number of steps the inputs will not change, the most recent input bits will be equal to the respective current inputs, there will be at least one vertex with leader bit equal to 1, and all leaders (and all vertices) will have positive counters, or they will all have zero counters, or they will all have negative counters. Thus, the outputs stabilize to 0 or 1 in any such computation.

4.3 Simulating the all-pairs graph with stabilizing inputs

In this section, we generalize a theorem of [2] to prove that any predicate stably computable with stabilizing inputs in the all-pairs family of graphs can be stably computed with stabilizing inputs in the unrestricted family of graphs. We use nondeterminism to simplify the description of the simulator.

Theorem 8 *For any protocol $\mathcal{A}$ there exists a protocol $\mathcal{B}$ such that for every n, if $\mathcal{A}$ stably computes predicate P with stabilizing inputs in the all-pairs interaction graph with n vertices and G is any*

communication graph with n vertices, protocol $\mathcal{B}$ stably computes predicate P with stabilizing inputs in G.

Proof sketch: Let $\mathcal{A}$ have states Q, initial state q_0, inputs X, outputs Y, and transition function δ. If we did not have to deal with stabilizing inputs, we could construct a very simple nondeterministic protocol to simulate $\mathcal{A}$ using the states Q, choosing at each interaction whether to exchange the states of the two vertices (which eventually allows any pair of simulated vertices to interact) or to take a step of the transition function δ, simulating a step of $\mathcal{A}$. With stabilizing inputs, we have to compensate for the fact that the simulated states move around the graph, but the inputs are supplied at fixed vertices.

Details are left to the full paper.

∎

Corollary 9 *The Presburger predicates are stably computable with stabilizing inputs in the unrestricted family of graphs.*

Since the predicate "no input is Z" is Presburger, this corollary completes the proof of Lemma 3.

5 Computing in bounded-degree networks

In Section 3 we showed that there is a protocol that stably computes whether G is a directed cycle. If G is a directed cycle, another protocol can organize a Turing machine computation of space $O(n)$ in the graph to determine properties of the inputs. Thus, certain graph structures can be recognized and exploited to give very powerful computational capabilities.

In this section we generalize this result to the family of graphs with degree at most d. A straightforward generalization of the protocol in Lemma 2 does not work, as we now illustrate. Consider the protocol below, which attempts to use the states R_1 and R_2 to keep track of the in-degree of a vertex, with state Y intended to indicate that a vertex of in-degree greater than 2 has been detected.

$$\begin{array}{lllll}
(1) & (-,-) & \mapsto & (I_1,R_1)\\
(2) & (I_1,R_1) & \mapsto & (-,-)\\
(3) & (-,R_1) & \mapsto & (I_2,R_2)\\
(4) & (I_2,R_2) & \mapsto & (-,R_1)\\
(5) & (-,R_2) & \mapsto & (Y,Y)\\
(6) & (Y,x) & \mapsto & (Y,Y)\\
(7) & (x,Y) & \mapsto & (Y,Y)
\end{array}$$

This protocol fails on the graph G with five vertices and edges $(1,2),(3,2),(3,4),(5,4)$, as witnessed by the computation:

$$(-,-,-,-,-),(-,R_1,I_1,-,-),(-,R_1,I_1,R_1,I_1),(-,R_1,-,-,I_1),(-,R_2,I_2,-,I_1),(Y,Y,I_2,-,I_1).$$

However, a more complex protocol using a leader works.

Lemma 10 *For every positive integer d, there is a protocol that stably computes whether G has maximum degree less than or equal to d.*

Proof sketch: We describe a protocol to determine whether G has any vertex of out-degree greater than d; in-degree is analogous, and the "nor" of these properties is what is required. The protocol is nondeterministic and based on leader election. Every vertex is initially a leader and has output 0. Each leader repeatedly engages in the following searching behavior. It hops around the vertices of G and decides to check a vertex by marking it and attempting to place $d+1$ markers of a second type at the ends of edges outgoing from the vertex being checked. It may decide to stop checking, collect a set of markers corresponding to the ones it set out, and resume its searching behavior. If it succeeds in placing all its markers, it changes its output to 1, hops around the graph to collect a set of markers corresponding to the ones it set out, and resumes its searching behavior.

When two leaders meet, one becomes a non-leader, and the remaining leader collects a set of markers corresponding to the ones that both had set out, changes its output back to 0, and resumes its searching behavior. Non-leaders copy the output of any leader they meet.

After the leader becomes unique and collects the set of markers that it and the last deposed leader had set out, then the graph is clear of markers and the unique leader resumes the searching behavior with its output set to 0. If there is no vertex of out-degree greater than d, the output will remain 0 (and will eventually be copied by all the non-leaders.) If there is some vertex of out-degree greater than d, the searching behavior eventually finds it and sets the output to 1, which will eventually be copied by all the non-leaders. ∎

Note that this technique can be generalized (using a finite collection of distinguishable markers) to determine the existence of any fixed subgraph in G. (For the lemma, the fixed subgraph is the out-directed star on $d+2$ vertices.) Another variant of this idea (mark a vertex and try to reach the mark by following directed edges) gives protocols to determine whether G contains a directed cycle or a directed odd cycle.

Theorem 11 *There are protocols that stably compute whether G contains a fixed subgraph, or a directed cycle, or a directed cycle of odd length in the unrestricted family of graphs.*

For bounded-degree graphs, we can organize the nodes into a spanning tree rooted at a leader, which can then distributed a Turing machine tape of size $O(n)$ across the nodes. This allows a population with a bounded-degree interaction graph to compute any function of the graph structure that is computable in linear space.

Theorem 12 *For every positive integer d, there is a protocol that for any d-bounded graph G stably constructs a spanning tree structure in G that can be used to simulate n cells of Turing machine tape.*

Proof sketch: The protocol is rather involved; details are deferred to the full paper. We give an outline of the protocol here.

The starting point is to label the vertices of G in such a way that no two neighbors of any vertex have the same label. A vertex can then send messages to a specific neighbor by waiting to encounter another vertex with the appropriate label.

To see that such a labeling exists, consider the graph G' obtained from G by ignoring the direction of edges and including an edge between any two nodes at distance 2 in G. A proper coloring of G' will give the required labeling of G, and the degree of G' is at most $4d^2$, so G' can be colored with $4d^2 + 1$ colors.

One part of the protocol eventually constructs a labeling of the desired kind by using leader election and a searching behavior that attempts to find two vertices at distance two that have the same label and nondeterministically relabel both. Eventually there will be no more relabeling.

The other part of the protocol attempts to use the constructed labeling to build a spanning tree structure in G. This is also based on leader election. Each leader begins building a spanning tree from the root, recording in each vertex the labels of the known neighbors of the vertex, and designating (by label) a parent for each non-root vertex. The leader repeatedly traverses its spanning tree attempting to recruit new vertices. When it meets another leader, one becomes a non-leader and the other begins building a new spanning tree from scratch.

The labeling portion of the protocol is running in parallel with the tree-construction, and it produces a "restart" marker whenever it relabels vertices. When a leader encounters a "restart" marker, it deletes the marker and again begins building a new spanning tree from scratch.

Eventually all the relabeling will be completed, and only one leader will remain, and the final spanning tree construction will not be restarted. However, it is important that the spanning tree construction be able to succeed given a correct labeling but otherwise arbitrary states left over from preceding spanning tree construction attempts. The leader repeatedly traverses the constructed spanning tree, setting a phase indicator (0 or 1) at each pass to detect vertices that are not yet part of the tree. An arbitrary ordering on the labels gives a fixed traversal order for the spanning tree, and a portion of each state can be devoted to simulating a Turing machine tape cell.

Because a leader cannot determine when it is unique, when the labeling is stable, or when the spanning tree is complete, it simply restarts computation in the tree each time it begins rebuilding a spanning tree. Eventually, however, the computation has access to n simulated cells of tape. ∎

Itkis and Levin use a similar construction in their self-stabilizing protocols for identical nameless nodes in an asynchronous communication network (represented by an undirected, connected, reflexive graph G) with worst-case transient faults [16]. In their model, in addition to a finite number of bits of storage, each node x maintains a finite set of pointers to itself and its neighbors, and can detect whether a neighbor y has a pointer to x and/or y, and can set a pointer to point to the first (in a fixed ordering) neighbor with a given property. This additional information about the graph structure permits them to exploit storage $O(|V|)$ in all cases. By comparison, in our model there are no pointers and each device truly has only a constant amount of memory regardless of the size and topology of the network. For this reason, in an all-pairs graph of size n, only memory proportional to $\log n$ is achievable. So the bounded-degree restriction of Theorem 12, or some other limitation that excludes the all-pairs graph, appears to be necessary.

6 Discussion and open problems

It is open whether every predicate stably computable with unchanging inputs in a family of graphs is stably computable with stabilizing inputs in the same family of graphs. For the family of all-

pairs graphs, both classes contain the Presburger predicates. It follows that if, in this family, some predicate is stably computable with unchanging inputs but not with stabilizing inputs, that predicate would not be Presburger. The existence of such a predicate would disprove a conjecture from [2].

A natural measure of the information-theoretic memory capacity of a graph G with a finite set of vertex labels L is the log of number of different isomorphism classes of G with vertex labels drawn from L. If L has at least two elements, the all-pairs graphs have memory capacity $\Theta(\log n)$, which can be exploited for computation in the setting of randomized interactions and computation with errors [2]. When the cardinality of L is large enough compared to d ($O(d^2)$ suffices), the memory capacity of d-degree bounded graphs is $\Theta(n)$, which we have shown above can be exploited by protocols for stable computation. Are there protocols to exploit the full-information theoretic memory capacity of arbitrary network communication graphs?

An important future direction is to study the running time of protocols in this model, perhaps under a stochastic model of pairwise interactions, as in the model of conjugating automata [2].

7 Acknowledgments

The authors would like to thank David Eisenstat for helpful comments on the paper.

References

[1] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Urn automata. Technical Report YALEU/DCS/TR–1280, Yale University Department of Computer Science, Nov. 2003.

[2] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 290–299. ACM Press, 2004.

[3] L. Bernardinello and F. D. Cindio. A survey of basic net models and modular net classes. In G. Rozenberg, editor, *Advances in Petri Nets: The DEMON Project*, volume 609 of *Lecture Notes in Computer Science*, pages 304–351. Springer-Verlag, 1992.

[4] G. Berry and G. Boudol. The Chemical Abstract Machine. *Theoretical Computer Science*, 96:217–248, 1992.

[5] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, Apr. 1983.

[6] I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis. Smart dust protocols for local detection and propagation. In *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 9–16. ACM Press, 2002.

[7] Z. Diamadi and M. J. Fischer. A simple game for the study of trust in distributed systems. *Wuhan University Journal of Natural Sciences*, 6(1–2):72–82, Mar. 2001. Also appears as Yale Technical Report TR–1207, January 2001, available at URL `ftp://ftp.cs.yale.edu/pub/TR/tr1207.ps`.

[8] J. Esparza. Decidability and complexity of Petri net problems-an introduction. In G. Rozenberg and W. Reisig, editors, *Lectures on Petri Nets I: Basic models.*, pages 374–428. Springer Verlag, 1998. Published as LNCS 1491.

[9] J. Esparza and M. Nielsen. Decibility issues for Petri nets - a survey. *Journal of Informatik Processing and Cybernetics*, 30(3):143–160, 1994.

[10] Q. Fang, F. Zhao, and L. Guibas. Lightweight sensing and communication protocols for target enumeration and aggregation. In *Proceedings of the 4th ACM International Symposium on Mobile ad hoc networking & computing*, pages 165–176. ACM Press, 2003.

[11] S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

[12] M. Grossglauser and D. N. C. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Transactions on Networking*, 10(4):477–486, 2002.

[13] J. Hopcroft and J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1978.

[14] O. H. Ibarra, Z. Dang, and O. Egecioglu. Catalytic p systems, semilinear sets, and vector addition systems. *Theor. Comput. Sci.*, 312(2-3):379–399, 2004.

[15] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile computing and networking*, pages 56–67. ACM Press, 2000.

[16] G. Itkis and L. A. Levin. Fast and lean self-stabilizing asynchronous protocols. In *Proceeding of 35th Annual Symposium on Foundations of Computer Science*, pages 226–239. IEEE Press, 1994.

[17] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for "Smart Dust". In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 271–278. ACM Press, 1999.

[18] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. In OSDI 2002: Fifth Symposium on Operating Systems Design and Implementation, December, 2002.

[19] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

[20] R. Milner. Bigraphical reactive systems: basic theory. Technical report, University of Cambridge, 2001. UCAM-CL-TR-523.

[21] B. Thorstensen, T. Syversen, T.-A. Bjornvold, and T. Walseth. Electronic shepherd: a low-cost, low-bandwidth, wireless network system. In *MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 245–255. ACM Press, 2004.

[22] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich. Collaborative signal and information processing: An information directed approach. *Proceedings of the IEEE,* 91(8):1199–1209, 2003.