Hardness of Computing and Approximating Predicates and Functions with Leaderless Population Protocols*

Amanda Belleville[1], David Doty[2], and David Soloveichik[3]

1 **Computer Science, University of California, Davis, CA, USA**
acbelleville@ucdavis.edu

2 **Computer Science, University of California, Davis, CA, USA**
doty@ucdavis.edu

3 **Electrical and Computer Engineering, University of Texas, Austin, TX, USA**
david.soloveichik@utexas.edu

Abstract

Population protocols are a distributed computing model appropriate for describing massive numbers of agents with very limited computational power (finite automata in this paper), such as sensor networks or programmable chemical reaction networks in synthetic biology. A population protocol is said to require a leader if every valid initial configuration contains a single agent in a special "leader" state that helps to coordinate the computation. Although the class of predicates and functions computable with probability 1 (stable computation) is the same whether a leader is required or not (semilinear functions and predicates), it is not known whether a leader is necessary for fast computation. Due to the large number of agents n (synthetic molecular systems routinely have trillions of molecules), efficient population protocols are generally defined as those computing in polylogarithmic in n (parallel) time. We consider population protocols that start in leaderless initial configurations, and the computation is regarded finished when the population protocol reaches a configuration from which a different output is no longer reachable.

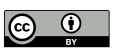
In this setting we show that a wide class of functions and predicates computable by population protocols are not *efficiently* computable (they require at least linear time), nor are some linear functions even efficiently *approximable*. It requires at least linear time for a population protocol even to approximate division by a constant or subtraction (or any linear function with a coefficient outside of $\mathbb{N}$), in the sense that for sufficiently small $\gamma > 0$, the output of a sublinear time protocol can stabilize outside the interval $f(m)(1 \pm \gamma)$ on infinitely many inputs m. In a complementary positive result, we show that with a sufficiently large value of γ, a population protocol *can* approximate any linear f with nonnegative rational coefficients, within approximation factor γ, in $O(\log n)$ time. We also show that it requires linear time to exactly compute a wide range of semilinear functions (e.g., $f(m) = m$ if m is even and $2m$ if m is odd) and predicates (e.g., parity, equality).

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases population protocol, time lower bound, stable computation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.141

* The first two authors were supported by NSF grant 1619343 and the third author by NSF grant 1618895.



44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 141; pp. 141:1–141:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Population protocols were introduced by Angluin, Aspnes, Diamadi, Fischer, and Peralta[3] as a model of distributed computing in which the agents have very little computational power and no control over their schedule of interaction with other agents. They can be thought of as a special case of a model of concurrent processing introduced in the 1960s, known alternately as vector addition systems[16], Petri nets[19], or commutative semi-Thue systems (or, when all transitions are reversible, "commutative semigroups")[9, 17]. As well as being an appropriate model for electronic computing scenarios such as sensor networks, they are a useful abstraction of "fast-mixing" physical systems such as animal populations[22], gene regulatory networks[8], and chemical reactions.

The latter application is especially germane: several recent wet-lab experiments demonstrate the systematic engineering of custom-designed chemical reactions [23, 12, 7, 20], unfortunately in all cases having a cost that scales linearly with the number of unique chemical species (states). (The cost can even be quadratic if certain error-tolerance mechanisms are employed [21].) Thus, it is imperative in implementing a molecular computational system to keep the number of distinct chemical species at a minimum. On the other hand, it is common (and relatively cheap) for the total number of such molecules (agents) to number in the trillions in a single test tube. It is thus important to understand the computational power enabled by a large number of agents n, where each agent has only a constant number of states (each agent is a finite state machine).

A population protocol is said to require a leader if every valid initial configuration contains a single agent in a special "leader" state that helps to coordinate the computation. Studying computation without a leader is important for understanding essentially distributed systems where symmetry breaking is difficult. Further, in the chemical setting obtaining single-molecule precision in the initial configuration is difficult. Thus, it would be highly desirable if the population protocol did not require an exquisitely tuned initial configuration.

1.1 Introduction to the model

A population protocol is defined by a finite set Λ of *states* that each agent may have, together with a *transition function*[1] $\delta : \Lambda^2 \to \Lambda^2$. A *configuration* is a nonzero vector $\mathbf{c} \in \mathbb{N}^\Lambda$ describing, for each $\overline{s} \in \Lambda$, the *count* $\mathbf{c}(\overline{s})$ of how many agents are in state $\overline{s}$. By convention we denote the number of agents by $n = \|\mathbf{c}\| = \sum_{\overline{s}\in\Lambda} \mathbf{c}(\overline{s})$. Given states $\overline{r}_1, \overline{r}_2, \overline{p}_1, \overline{p}_2 \in \Lambda$, if $\delta(\overline{r}_1, \overline{r}_2) = (\overline{p}_1, \overline{p}_2)$ (denoted $\overline{r}_1, \overline{r}_2 \to \overline{p}_1, \overline{p}_2$), and if a pair of agents in respective states $\overline{r}_1$ and $\overline{r}_2$ interact, then their states become $\overline{p}_1$ and $\overline{p}_2$.[2] The next pair of agents to interact is chosen uniformly at random. The expected (parallel) time for any event to occur is the expected number of interactions, divided by the number of agents n. This measure of time is based on the natural parallel model where each agent participates in a constant number of interactions in one unit of time; hence $\Theta(n)$ total interactions are expected per unit time [5].

The most well-studied population protocol task is computing Boolean-valued *predicates*. It is known that a protocol *stably decides* a predicate $\phi : \mathbb{N}^k \to \{0, 1\}$ (meaning computes

[1] Some work allows nondeterministic transitions, in which the transition function maps to subsets of $\Lambda \times \Lambda$. Our results are independent of whether transitions are nondeterministic, and we choose a deterministic, symmetric transition function, rather than a more general relation $\delta \subseteq \Lambda^4$, merely for notational convenience.

[2] In the most generic model, there is no restriction on which agents are permitted to interact. If one prefers to think of the agents as existing on nodes of a graph, then it is the complete graph K_n for a population of n agents.

the correct answer with probability 1; see Section 6 for a formal definition) if [3] and only if [4] ϕ is semilinear.

Population protocols can also compute integer-valued *functions* $f : \mathbb{N}^k \to \mathbb{N}$. Suppose we start with $m \leq n/2$ agents in "input" state $\overline{x}$ and the remaining agents in a "quiescent" state $\overline{q}$. Consider the protocol with a single transition rule $\overline{x}, \overline{q} \to \overline{y}, \overline{y}$. Eventually exactly $2m$ agents are in the "output" state $\overline{y}$, so this protocol computes the function $f(m) = 2m$. Furthermore (letting $\#\overline{s}$ = count of state $\overline{s}$), if $\#\overline{q} - 2m = \Omega(n)$ initially (e.g., $\#\overline{q} = 3m$), then it takes $\Theta(\log n)$ expected time until $\#\overline{y} = 2m$. Similarly, the transition rule $\overline{x}, \overline{x} \to \overline{y}, \overline{q}$ computes the function $f(m) = \lfloor m/2 \rfloor$, but exponentially slower, in expected time $\Theta(n)$. The transitions $\overline{x}_1, \overline{q} \to \overline{y}, \overline{q}$ and $\overline{x}_2, \overline{y} \to \overline{q}, \overline{q}$ compute $f(m_1, m_2) = m_1 - m_2$ (assuming $m_1 \geq m_2$), also in time $\Theta(n)$ if $m_1 = m_2 + O(1)$.

Formally, we say a population protocol *stably computes* a function $f : \mathbb{N}^k \to \mathbb{N}$ if, for every "valid" initial configuration $\mathbf{i} \in \mathbb{N}^\Lambda$ representing input $\mathbf{m} \in \mathbb{N}^k$ (via counts $\mathbf{i}(\overline{x}_1), \ldots, \mathbf{i}(\overline{x}_k)$ of "input" states $\Sigma = \{\overline{x}_1, \ldots, \overline{x}_k\} \subseteq \Lambda$) with probability 1 the system reaches from $\mathbf{i}$ to $\mathbf{o}$ such that $\mathbf{o}(\overline{y}) = f(\mathbf{m})$ ($\overline{y} \in \Lambda$ is the "output" state) and $\mathbf{o}'(\overline{y}) = \mathbf{o}(\overline{y})$ for every $\mathbf{o}'$ reachable from $\mathbf{o}$ (i.e., $\mathbf{o}$ is *stable*). Defining what constitutes a "valid" initial configuration (i.e., what non-input states can be present initially, and how many) is nontrivial. In this paper we focus on population protocols without a *leader*—a state present in count 1, or small count—in the initial configuration. Here, we equate "leaderless" with initial configurations in which no positive state count is sublinear in the population size n.

It is known that a function $f : \mathbb{N}^k \to \mathbb{N}$ is stably computable by a population protocol if and only if its *graph* $\{(\mathbf{m}, f(\mathbf{m})) \mid \mathbf{m} \in \mathbb{N}^k\} \subset \mathbb{N}^{k+1}$ is a semilinear set [4, 11]. This means intuitively that it is piecewise affine, with each affine piece having rational slopes.

Despite the exact characterization of predicates and functions stably computable by population protocols, we still lack a full understanding of which of the stably computable (i.e., semilinear) predicates and functions are computable *quickly* (say, in time polylogarithmic in n) and which are only computable slowly (linear in n). For positive results, significantly more is known about time to *convergence* [5] with a leader (time to reach a configuration with the correct answer). In this paper we shed new light on time to *stabilization* without a leader (time to reach a configuration from which the answer is *guaranteed to remain correct*).

1.2 Contributions

Definition of function computation and approximation. We formally define computation and approximation of functions $f : \mathbb{N}^k \to \mathbb{N}$ for population protocols. This mode of computation was discussed briefly in the first population protocols paper[3, Section 3.4], which focused more on Boolean predicate computation, and it was defined formally in the more general model of chemical reaction networks[11, 13]. Some subtle issues arise that are unique to population protocols. We also formally define a notion of function *approximation* with population protocols, which has its own issues.

Inapproximability of most linear functions with sublinear time and sublinear error. Recall that the transition rule $\overline{x}, \overline{x} \to \overline{y}, \overline{q}$ computes $f(m) = \lfloor m/2 \rfloor$ in linear time. Consider the transitions $\overline{a}, \overline{x} \to \overline{b}, \overline{y}$ and $\overline{b}, \overline{x} \to \overline{a}, \overline{q}$, starting with $\#\overline{x} = m$, $\#\overline{a} = \gamma m$ for some $0 < \gamma < 1$, and $\#\overline{y} = \#\overline{q} = 0$ (so $n = m + \gamma m$ total agents). Then eventually $\#\overline{y} \in \{m/2, \ldots, m/2 + \gamma m\}$ and $\#\overline{x} = 0$ (stabilizing $\#\overline{y}$), after $O(\frac{1}{\gamma} \log n)$ expected time. (This is analyzed in more detail in Section 5.) Thus, if we tolerate an error linear in n, then f can be approximated in logarithmic time. However, Theorem 4.1 shows this error bound to be tight: *any* leaderless population protocol that approximates $f(m) = \lfloor m/2 \rfloor$, or any other linear function with a

coefficient outside of $\mathbb{N}$ (such as $\lfloor 4m/3 \rfloor$ or $m_1 - m_2$), requires at least linear time to achieve sublinear error.

As a corollary, such functions cannot be stably computed in sublinear time (since computing exactly is the same as approximating with zero error). Conversely, it is simple to show that any linear function with all coefficients in $\mathbb{N}$ is stably computable in logarithmic time (Observation 5.1). Thus we have a dichotomy theorem for the efficiency (with regard to stabilization) of computing linear functions f by leaderless population protocols: if all of f's coefficients are in $\mathbb{N}$, then it is computable in logarithmic time, and otherwise it requires linear time.

Approximability of nonnegative rational-coefficient linear functions with logarithmic time and linear error. Theorem 4.1 says that no linear function with a coefficient outside of $\mathbb{N}$ can be stably computed with sublinear time and sublinear error. In a complementary positive result, Theorem 5.2, by relaxing the error to linear, and restricting the coefficients to be *nonnegative* rationals (but not necessarily integers), we show how to approximate any such linear function in logarithmic time. (It is open if $m_1 - m_2$ can be approximated with linear error in logarithmic time.)

Uncomputability of many nonlinear functions in sublinear time. What about non-linear functions? Theorem 3.1 states that sublinear time computation cannot go much beyond linear functions with coefficients in $\mathbb{N}$. We show any function computable in sublinear time is *eventually*-$\mathbb{N}$-*linear*, which we define to be linear with nonnegative integer coefficients on all sufficiently large inputs. Examples of non-eventually-$\mathbb{N}$-linear functions, that provably cannot be computed in sublinear time, include $f(m_1, m_2) = \min(m_1, m_2)$ (computable slowly via $\overline{x}_1, \overline{x}_2 \to \overline{y}, \overline{q}$), and $f(m) = m - 1$ (computable slowly via $\overline{x}, \overline{x} \to \overline{x}, \overline{y}$).

The only remaining semilinear functions whose asymptotic time complexity remains unknown are those "piecewise linear" functions that switch between pieces only near the boundary of $\mathbb{N}^k$; for example, $f(m) = 0$ if $m \leq 3$ and $f(m) = m$ otherwise.

Undecidability of many predicates in sublinear time. Every semilinear predicate $\phi : \mathbb{N}^k \to \{0, 1\}$ is stably decidable in $O(n)$ time [5]. Some, such as $\phi(m) = 1$ iff $m \geq 1$, are stably decidable in $O(\log n)$ time by a leaderless protocol, in this case by the transition $\overline{x}, \overline{q} \to \overline{x}, \overline{x}$, where $\overline{x}$ "votes" for output 1 and $\overline{q}$ votes 0. A predicate is *eventually constant* if $\phi(\mathbf{m}_0) = \phi(\mathbf{m}_1)$ for all sufficiently large $\mathbf{m}_0, \mathbf{m}_1$. We show that if a leaderless population protocol stably decides a predicate ϕ in sublinear time, then ϕ is eventually constant. Examples of non-eventually constant predicates include parity ($\phi(m) = 1$ iff m is odd), majority ($\phi(m_1, m_2) = 1$ iff $m_1 \geq m_2$), and equality ($\phi(m_1, m_2) = 1$ iff $m_1 = m_2$). It does *not* include certain semilinear predicates, such as $\phi(m) = 1$ iff $m \geq 1$ (decidable in $O(\log n)$ time) or $\phi(m) = 1$ iff $m \geq 2$ (decidable in $O(n)$ time, and no faster protocol is known).

Note that there is a fundamental difficulty in extending the last two stated negative results to functions and predicates that "do something different only near the boundary of $\mathbb{N}^k$". This is because for inputs where one state is present in small count, the population protocol could in principle use that input as a "leader state"—and no longer be leaderless.

It is possible that the non-eventually constant predicates and non-eventually-$\mathbb{N}$-linear functions, which cannot be computed in sublinear time in our setting, *could* be efficiently computed in the following ways: (1) *With an initial leader* stabilizing to the correct answer in sublinear time, (2) Without initial leaders but *converging* to the correct output in sublinear time. (3) (With or without a leader) stabilizing to an output in sublinear time but *allowing a small probability of incorrect output.*

1.3 Related work

Positive results. Angluin, Aspnes, Diamadi, Fischer, and Peralta [3] showed that any semilinear predicate can be decided in expected parallel time $O(n \log n)$, later improved to $O(n)$ by Angluin, Aspnes, and Eisenstat [5]. More strikingly, the latter paper showed that if an initial leader is present (a state assigned to only a single agent in every valid initial configuration), then there is a protocol for ϕ that *converges* to the correct answer in expected time $O(\log^5 n)$. However, this protocol's expected time to *stabilize* is still provably $\Omega(n)$. Chen, Doty, and Soloveichik [11] showed in the related model of chemical reaction networks (borrowing techniques from the related predicate results [3, 4]) that any semilinear *function* (integer-output $f : \mathbb{N}^k \to \mathbb{N}$) can similarly be computed with expected convergence time $O(\log^5 n)$ if an initial leader is present, but again with much slower stabilization time $O(n \log n)$. Doty and Hajiaghayi [13] showed that any semilinear function can be computed by a chemical reaction network without a leader with expected convergence and stabilization time $O(n)$. Although the chemical reaction network model is more general, these results hold for population protocols.

Since efficient computation seems to be helped by a leader, the computational task of leader election has received significant recent attention. In particular, Alistarh and Gelashvili [2] showed that in a variant of the model allowing the number of states λ_n to grow with the population size n, a protocol with $\lambda_n = O(\log^3 n)$ states can elect a leader with high probability in $O(\log^3 n)$ expected time. Alistarh, Aspnes, Eisenstat, Gelashvili, and Rivest [1] later showed how to reduce the number of states to $\lambda_n = O(\log^2 n)$, at the cost of increasing the expected time to $O(\log^{5.3} n \log \log n)$.

Negative results. The first attempt to show the limitations of sublinear time population protocols, using the more general model of chemical reaction networks, was made by Chen, Cummings, Doty, and Soloveichik [10]. They studied a variant of the problem in which negative results are easier to prove, an "adversarial worst-case" notion of sublinear time: the protocol is required to be sublinear time not only from the initial configuration, but also from any reachable configuration. They showed that the predicates computable in this manner are precisely those whose output depends only on the presence or absence of states (and not on their exact positive counts). Doty and Soloveichik [14] showed the first $\Omega(n)$ lower bound on expected time from valid initial configurations, proving that any protocol electing a leader with probability 1 takes $\Omega(n)$ time.

These techniques were recently improved by Alistarh, Aspnes, Eisenstat, Gelashvili, and Rivest [1], who showed that even with up to $\lambda_n = O(\log \log n)$ states, any protocol electing a leader with probability 1 requires nearly linear time: $\Omega(n/\text{polylog } n)$. They used these tools to prove time lower bounds for another important computational task: majority (detecting whether state $\overline{x}_1$ or $\overline{x}_2$ is more numerous in the initial population, by stabilizing on a configuration in which the state with the larger initial count occupies the whole population).

In contrast to these previous results on the specific tasks of leader election and majority, we obtain time lower bounds for a broad class of functions and predicates, showing "most" of those computable at all by population protocols, cannot be computed in sublinear time. Since they all *can* be computed in linear time, this settles their asymptotic population protocol time complexity.

Informally, one explanation for our result could be that some computation requires electing "leaders" as part of the computation, and other computation does not. Since leader election itself requires linear time as shown in [14], the computation that requires it is necessarily inefficient. It is not clear, however, how to define the notion of a predicate or function

computation requiring electing a leader somewhere in the computation, but recent work by Michail and Spirakis helps to clarify the picture [18].

2 Preliminaries

If Λ is a finite set (in this paper, of *states*, which will be denoted as lowercase Roman letters with an overbar such as $\overline{s}$), we write $\mathbb{N}^\Lambda$ to denote the set of functions $\mathbf{c} : \Lambda \to \mathbb{N}$. Equivalently, we view an element $\mathbf{c} \in \mathbb{N}^\Lambda$ as a vector of $|\Lambda|$ nonnegative integers, with each coordinate "labeled" by an element of Λ. (By assuming some canonical ordering $\overline{s}_1, \ldots, \overline{s}_k$ of Λ, we also interpret $\mathbf{c} \in \mathbb{N}^\Lambda$ as a vector $\mathbf{c} \in \mathbb{N}^k$.) Given $\overline{s} \in \Lambda$ and $\mathbf{c} \in \mathbb{N}^\Lambda$, we refer to $\mathbf{c}(\overline{s})$ as the *count of* $\overline{s}$ *in* $\mathbf{c}$. Let $\|\mathbf{c}\| = \|\mathbf{c}\|_1 = \sum_{\overline{s}\in\Lambda} \mathbf{c}(\overline{s})$. We write $\mathbf{c} \leq \mathbf{c}'$ to denote that $\mathbf{c}(\overline{s}) \leq \mathbf{c}'(\overline{s})$ for all $\overline{s} \in \Lambda$. Since we view vectors $\mathbf{c} \in \mathbb{N}^\Lambda$ equivalently as multisets of elements from Λ, if $\mathbf{c} \leq \mathbf{c}'$ we say $\mathbf{c}$ is a *subset* of $\mathbf{c}'$. For $\alpha > 0$, we say that $\mathbf{c} \in \mathbb{N}^k$ is α-*dense* if, for all $i \in \{1, \ldots, k\}$, if $\mathbf{c}(i) > 0$, then $\mathbf{c}(i) \geq \alpha\|\mathbf{c}\|$.

It is sometimes convenient to use multiset notation to denote vectors, e.g., $\{\overline{x}, \overline{x}, \overline{y}\}$ and $\{2\overline{x}, \overline{y}\}$ both denote the vector $\mathbf{c}$ defined by $\mathbf{c}(\overline{x}) = 2$, $\mathbf{c}(\overline{y}) = 1$, and $\mathbf{c}(\overline{s}) = 0$ for all $\overline{s} \notin \{\overline{x}, \overline{y}\}$. Given $\mathbf{c}, \mathbf{c}' \in \mathbb{N}^\Lambda$, we define the vector component-wise operations of addition $\mathbf{c} + \mathbf{c}'$, subtraction $\mathbf{c} - \mathbf{c}'$, and scalar multiplication $m\mathbf{c}$ for $m \in \mathbb{N}$. For a set $\Delta \subset \Lambda$, we view a vector $\mathbf{c} \in \mathbb{N}^\Delta$ equivalently as a vector $\mathbf{c} \in \mathbb{N}^\Lambda$ by assuming $\mathbf{c}(\overline{s}) = 0$ for all $\overline{s} \in \Lambda \setminus \Delta$. Write $\mathbf{c} \restriction \Delta$ to denote the vector $\mathbf{d} \in \mathbb{N}^\Delta$ such that $\mathbf{c}(\overline{s}) = \mathbf{d}(\overline{s})$ for all $\overline{s} \in \Delta$. In this paper, the floor function $\lfloor\cdot\rfloor : \mathbb{R} \to \mathbb{Z}$ is defined to be the integer *closest to 0* that is distance < 1 from the input, e.g., $\lfloor -3.4 \rfloor = -3$ and $\lfloor 3.4 \rfloor = 3$.

We say a function $f : \mathbb{N}^k \to \mathbb{N}$ is *eventually-*$\mathbb{N}$*-affine* if there are $b, c_1, \ldots, c_k \in \mathbb{N}$ and $m_0 \in \mathbb{N}$ such that for all $\mathbf{m} \in \mathbb{N}^k_{\geq m_0}$, $f(\mathbf{m}) = b + \sum_{i=1}^k c_i \mathbf{m}(i)$. We say a function $f : \mathbb{N}^k \to \mathbb{N}$ is *eventually-*$\mathbb{N}$*-linear* if it is eventually-$\mathbb{N}$-affine with offset $b = 0$, i.e., if $f(\mathbf{0}) = 0$. We say the function is $\mathbb{N}$*-linear* if it is eventually-$\mathbb{N}$-linear with $m_0 = 0$. Similarly, a function is $\mathbb{Q}_{\geq 0}$*-linear* if there are $c_1, \ldots, c_k \in \mathbb{Q}_{\geq 0}$ such that for all $\mathbf{m} \in \mathbb{N}^k$, $f(\mathbf{m}) = \sum_{i=1}^k \lfloor c_i \mathbf{m}(i) \rfloor$.

2.1 Population Protocols

A *population protocol* is a pair $\mathcal{P} = (\Lambda, \delta)$, where Λ is a finite set of *states* and $\delta : \Lambda^2 \to \Lambda^2$ is the (symmetric) *transition function*. A *configuration* of a population protocol is a vector $\mathbf{c} \in \mathbb{N}^\Lambda$, with the interpretation that $\mathbf{c}(\overline{s})$ agents are in state $\overline{s} \in \Lambda$. If there is some "current" configuration $\mathbf{c}$ understood from context, we write $\#\overline{s}$ to denote $\mathbf{c}(\overline{s})$. By convention, the value $n \in \mathbb{Z}_{\geq 1}$ represents the total number of agents $\|\mathbf{c}\|$. A *transition* is a 4-tuple $\tau = (\overline{r}_1, \overline{r}_2, \overline{p}_1, \overline{p}_2) \in \Lambda^4$, written $\tau : \overline{r}_1, \overline{r}_2 \to \overline{p}_1, \overline{p}_2$, such that $\delta(\overline{r}_1, \overline{r}_2) = (\overline{p}_1, \overline{p}_2)$. If an agent in state $\overline{r}_1$ interacts with an agent in state $\overline{r}_2$, then they change states to $\overline{p}_1$ and $\overline{p}_2$. This paper typically defines a protocol by a list of transitions, with δ implicit. There is a *null* transition $\delta(\overline{r}_1, \overline{r}_2) = (\overline{r}_1, \overline{r}_2)$ if a different output for $\delta(\overline{r}_1, \overline{r}_2)$ is not specified.

Given $\mathbf{c} \in \mathbb{N}^\Lambda$ and transition $\tau : \overline{r}_1, \overline{r}_2 \to \overline{p}_1, \overline{p}_2$, we say that τ is *applicable* to $\mathbf{c}$ if $\mathbf{c} \geq \{\overline{r}_1, \overline{r}_2\}$, i.e., $\mathbf{c}$ contains 2 agents, one in state $\overline{r}_1$ and one in state $\overline{r}_2$. If τ is applicable to $\mathbf{c}$, then write $\tau(\mathbf{c})$ to denote the configuration $\mathbf{c} - \{\overline{r}_1, \overline{r}_2\} + \{\overline{p}_1, \overline{p}_2\}$ (i.e., that results from applying τ to $\mathbf{c}$); otherwise $\tau(\mathbf{c})$ is undefined. A finite or infinite sequence of transitions (τ_i) is a *transition sequence*. Given a $\mathbf{c}_0 \in \mathbb{N}^\Lambda$ and a transition sequence (τ_i), the induced *execution sequence* (or *path*) is a finite or infinite sequence of configurations $(\mathbf{c}_0, \mathbf{c}_1, \ldots)$ such that, for all $i \geq 1$, $\mathbf{c}_i = \tau_{i-1}(\mathbf{c}_{i-1})$. If a finite execution sequence, with associated transition sequence $\overline{q}$, starts with $\mathbf{c}$ and ends with $\mathbf{c}'$, we write $\mathbf{c} \Longrightarrow_q \mathbf{c}'$. We write $\mathbf{c} \Longrightarrow_\mathcal{P} \mathbf{c}'$ (or $\mathbf{c} \Longrightarrow \mathbf{c}'$ when $\mathcal{P}$ is clear from context) if such a path exists (i.e., it is possible to reach from $\mathbf{c}$ to $\mathbf{c}'$) and we say that $\mathbf{c}'$ is *reachable* from $\mathbf{c}$. Let $\mathsf{post}_\mathcal{P}(\mathbf{c}) = \{\mathbf{c}' \mid \mathbf{c} \Longrightarrow_\mathcal{P} \mathbf{c}'\}$ to denote the set of

all configurations reachable from $\mathbf{c}$, writing $\mathsf{post}(\mathbf{c})$ when $\mathcal{P}$ is clear from context. If it is understood from context what is the initial configuration $\mathbf{i}$, then say $\mathbf{c}$ is simply *reachable* if $\mathbf{i} \Longrightarrow \mathbf{c}$. If a transition $\tau : \overline{r}_1, \overline{r}_2 \to \overline{p}_1, \overline{p}_2$ has the property that for $i \in \{1, 2\}$, $\overline{r}_i \notin \{\overline{p}_1, \overline{p}_2\}$, or if ($\overline{r}_1 = \overline{r}_2$ and ($\overline{r}_i \neq \overline{p}_1$ or $\overline{r}_i \neq \overline{p}_2$)), then we say that τ *consumes* $\overline{r}_i$; i.e., applying τ reduces the count of $\overline{r}_i$. We say τ *produces* $\overline{p}_i$ if it increases the count of $\overline{p}_i$.

2.2 Time Complexity

The model used to analyze time complexity is a discrete-time Markov process, whose states correspond to configurations of the population protocol. In any configuration the next interaction is chosen by selecting a pair of agents uniformly at random and applying transition function δ to determine the next configuration. Since a transition may be null, self-loops are allowed. To measure time we count the expected total number of interactions (including null), and divide by the number of agents n. (In the population protocols literature, this is often called "parallel time"; i.e. n interactions among a population of n agents corresponds to one unit of time). Let $\mathbf{c} \in \mathbb{N}^\Lambda$ and $C \subseteq \mathbb{N}^\Lambda$. Denote the probability that the protocol reaches from $\mathbf{c}$ to some configuration $\mathbf{c}' \in C$ by $\mathsf{Pr}[\mathbf{c} \Longrightarrow C]$. If $\mathsf{Pr}[\mathbf{c} \Longrightarrow C] = 1$, define the *expected time to reach from* $\mathbf{c}$ *to* C, denoted $\mathsf{T}[\mathbf{c} \Longrightarrow C]$, to be the expected number of interactions to reach from $\mathbf{c}$ to some $\mathbf{c}' \in C$, divided by the number of agents $n = \|\mathbf{c}\|$. If $\mathsf{Pr}[\mathbf{c} \Longrightarrow C] < 1$ then $\mathsf{T}[\mathbf{c} \Longrightarrow C] = \infty$.

3 Exact computation of nonlinear functions

In Section 4, we obtained a precise characterization of the linear functions stably computable in sublinear time by population protocols and furthermore show that those not exactly computable in sublinear time are not even approximable with sublinear error in sublinear time. However, the class of functions stably computable (in any amount of time) by population protocols is known to contain non-linear functions such as $f(m_1, m_2) = \max(m_1, m_2)$, or $f(m) = m$ if m is even and $f(m) = 2m$ if m is odd. In fact a function is stably computable by a population protocol if and only if its *graph* $\{(\mathbf{m}, f(\mathbf{m})) \mid \mathbf{m} \in \mathbb{N}^k\}$ is a semilinear set [4, 11]. A set $A \subseteq \mathbb{N}^k$ is *semilinear* if and only if [15] it is expressible as a finite number of unions, intersections, and complements of sets of one of the following two forms: *threshold sets* of the form $\{\mathbf{x} \mid \sum_{i=1}^k a_i \cdot \mathbf{x}(i) < b\}$ for some constants $a_1, \ldots, a_k, b \in \mathbb{Z}$ or *mod sets* of the form $\{\mathbf{x} \mid \sum_{i=1}^k a_i \cdot \mathbf{x}(i) \equiv b \mod c\}$ for some constants $a_1, \ldots, a_k, b, c \in \mathbb{N}$. Say that a set $P \subseteq \mathbb{N}^k$ is a *periodic coset* if there exist $\mathbf{b}, \mathbf{p}_1, \ldots, \mathbf{p}_l \in \mathbb{N}^k$ such that $P = \{\mathbf{b} + n_1\mathbf{p}_1 + \ldots + n_l\mathbf{p}_l \mid n_1, \ldots, n_l \in \mathbb{N}\}$. (These are typically called "linear" sets, but we wish to avoid confusion with linear functions.) Equivalently, a set is semilinear if and only if it is a finite union of periodic cosets. We say a function $f : \mathbb{N}^k \to \mathbb{N}$ is *semilinear* if its *graph* $\{(\mathbf{m}, f(\mathbf{m})) \mid \mathbf{m} \in \mathbb{N}^k\} \subset \mathbb{N}^{k+1}$ is a semilinear set. A function f is stably computable by a population protocol (given unbounded time) if and only if f is semilinear [11, 4].

Although our technique fails to completely characterize the efficient computability of all semilinear functions, we show that a wide class of semilinear functions cannot be stably computed in sublinear time: functions that are not eventually $\mathbb{N}$-linear. The only exceptions, for which we cannot prove linear time is required, yet neither is there known a counterexample protocol stably computing the function in sublinear time, are functions whose "non-integral-linearities are near the boundary of $\mathbb{N}^k$". For example, the function $f(m) = 0$ if $m \leq 3$ and $f(m) = m$ otherwise is non-linear (although it is semilinear, so stably computable), but restricted to the domain of inputs > 3, it is linear with positive integer coefficients. Thus it is an example of a function whose "population protocol time complexity" is unknown.

Corollary 4.2 and Observation 5.1 imply that a linear function is stably computable in sublinear time by a population protocol if and only if it is $\mathbb{N}$-linear. Theorem 3.1 generalizes the forward direction (restricted to nonlinear functions) to eventually-$\mathbb{N}$-linear functions.

We first give a formal definition of function computation by population protocols. A *function-computing population protocol* is a tuple $\mathcal{C} = (\Lambda, \delta, \Sigma, \overline{y}, \overline{q})$, where (Λ, δ) is a population protocol, $\Sigma = \{\overline{x}_1, \ldots, \overline{x}_k\} \subset \Lambda$ is the set of *input states*, $\overline{y} \in \Lambda$ is the *output state*, and $\overline{q} \in \Lambda \setminus \Sigma$ is the *quiescent state*. We say that a configuration $\mathbf{o} \in \mathbb{N}^\Lambda$ is *stable* if, for all $\mathbf{o}' \in \mathsf{post}(\mathbf{o})$, $\mathbf{o}(\overline{y}) = \mathbf{o}'(\overline{y})$, i.e., the count of $\overline{y}$ cannot change once $\mathbf{o}$ is reached.

Let $f : \mathbb{N}^k \to \mathbb{N}$, $\mathbf{i} \in \mathbb{N}^\Lambda$, and let $\mathbf{m} = \mathbf{i} \restriction \Sigma$. We say that $\mathcal{C}$ *stably computes* f *from* $\mathbf{i}$ if, for all $\mathbf{c} \in \mathsf{post}(\mathbf{i})$, there exists a stable $\mathbf{o} \in \mathsf{post}(\mathbf{c})$ such that $\mathbf{o}(\overline{y}) = f(\mathbf{m})$, i.e., $\mathcal{C}$ stabilizes to the correct output from the initial configuration $\mathbf{i}$. However, for any input $\mathbf{m} \in \mathbb{N}^k$, there are many initial configurations $\mathbf{i} \in \mathbb{N}^\Lambda$ representing it (i.e., such that $\mathbf{i} \restriction \Sigma = \mathbf{m}$). We now formalize what sort of initial configurations $\mathcal{C}$ is required to handle.

We say a function $q_0 : \mathbb{N}^k \to \mathbb{N}$ is *linearly bounded* if there is a constant $c \in \mathbb{N}$ such that, for all $\mathbf{m} \in \mathbb{N}^k$, $q_0(\mathbf{m}) \leq c\|\mathbf{m}\|$. We say that $\mathcal{C}$ *stably computes* f if there is a linearly bounded function $q_0 : \mathbb{N}^k \to \mathbb{N}$ such that, for any $\mathbf{i} \in \mathbb{N}^\Lambda$, defining $\mathbf{m} = \mathbf{i} \restriction \Sigma$, if $\mathbf{i}(\overline{q}) \geq q_0(\mathbf{m})$ and $\mathbf{i}(\overline{s}) = 0$ for all $\overline{s} \in \Lambda \setminus (\Sigma \cup \{\overline{q}\})$, then $\mathcal{C}$ stably computes f from $\mathbf{i}$. It is well-known[6] that this is equivalent to requiring, under the randomized model in which the next interaction is between a pair of agents picked uniformly at random, that the protocol stabilizes on the correct output with probability 1. More formally, given $f : \mathbb{N}^k \to \mathbb{N}$ and $\mathbf{m} \in \mathbb{N}^k$, defining $S^{\mathcal{C}}_{f,\mathbf{m}} = \{\mathbf{o} \in \mathbb{N}^\Lambda \mid \mathbf{o} \text{ is stable and } \mathbf{o}(\overline{y}) = f(\mathbf{m})\}$, $\mathcal{C}$ stably computes f if and only if, for all $\mathbf{m}$, defining $\mathbf{i}$ with $\mathbf{i} \restriction \Sigma = \mathbf{m}$ as above with $\mathbf{i}(\overline{q})$ sufficiently large, $\mathsf{Pr}\left[\mathbf{i} \Longrightarrow S^{\mathcal{C}}_{f,\mathbf{m}}\right] = 1$. It is also equivalent to requiring that every fair infinite execution leads to a correct stable configuration, where an execution is *fair* if every configuration infinitely often reachable appears infinitely often in the execution. We say that an initial configuration $\mathbf{i}$ so defined is *valid*. Since all semilinear functions are linearly bounded [11], a linearly bounded q_0 suffices to ensure there are enough agents to represent the output of a semilinear function, even if we choose $\mathbf{i}(\overline{q}) = q_0(\mathbf{i} \restriction \Sigma)$. If q_0 were *not* linearly bounded, and thus a super-linear count of state $\overline{q}$ is required, we would essentially need to do non-semilinear computation just to initialize the population protocol.

Let $f : \mathbb{N}^k \to \mathbb{N}$ and $t : \mathbb{N} \to \mathbb{N}$. Given a function-computing population protocol $\mathcal{C}$ that stably computes f, we say $\mathcal{C}$ *stably computes* f *in expected time* t if, for all valid initial configurations $\mathbf{i}$ of $\mathcal{C}$, letting $\mathbf{m} = \mathbf{i} \restriction \Sigma$, $\mathsf{T}\left[\mathbf{i} \Longrightarrow S^{\mathcal{C}}_{f,\mathbf{m}}\right] \leq t(n)$.

▶ **Theorem 3.1.** *Let $f : \mathbb{N}^k \to \mathbb{N}$, and let $\mathcal{C}$ be a function-computing population protocol that stably computes f. If f is not eventually-$\mathbb{N}$-linear then $\mathcal{C}$ takes expected time $\Omega(n)$.*

Techniques developed in previous work for proving time lower bounds [14, 1] can certainly generalize beyond leader election and majority, although it was not clear what precise category of computation they cover. However, to extend the impossibility results to all not eventually-$\mathbb{N}$-linear functions, we needed to develop new tools.

Both in prior and current work, the high level intuition of the proof technique is as follows. The overall argument is a proof by contradiction: if sublinear time computation is possible then we find a nefarious execution sequence which stabilizes to an incorrect output. In more detail, sublinear time computation requires avoiding "bottlenecks"—having to go through a transition in which both states are present in small count (constant independent of the number of agents n). Traversing even a single such transition requires linear time. Technical lemmas show that bottleneck-free execution sequences from α-dense initial configurations (i.e., initial configurations where every state that is present is present in at least αn count)

are amenable to predictable "surgery" [14, 1]. At the high level, the surgery lemmas show how states that are present in "low" count when the population protocol stabilizes, can be manipulated (added or removed) such that only "high" count other states are affected. Since it can also be shown that changing high count states in a stable configuration does not affect its stability, this means that the population protocol cannot "notice" the surgery, and remains stabilized to the previous output. For leader election, the surgery allows one to remove an additional leader state (leaving us with no leaders). For majority computation [1], the input in the minority must be present in low count (or absent) at the end. This allows one to add enough of the minority input to turn it into the majority, while the protocol continues to output the wrong answer.

However, applying the previously developed surgery lemmas to fool a more general function computing population protocol is more difficult. The surgery to consume additional input states affects the count of the output state, which could be present in "large count" at the end. How do we know that the effect of the surgery on the output is not consistent with the desired output of the function? In order to arrive at a contradiction we develop two new techniques, both of which are necessary to cover all cases. The first involves showing that the slope of the change in the count of the output state as a function of the input states is inconsistent. The second involves exposing the semilinear structure of the graph of the function being computed, and forcing it to enter the "wrong piece" (i.e., periodic coset).

4 Sublinear-time, sublinear-error approximation of linear functions with negative or non-integer coefficients is impossible

A *function-approximating population protocol* is a tuple $\mathcal{A} = (\Lambda, \delta, \Sigma, \overline{y}, \overline{q}, \overline{a})$, where $(\Lambda, \delta, \Sigma, \overline{y}, \overline{q})$ is a function-computing population protocol and $\overline{a} \in \Lambda \setminus (\Sigma \cup \{\overline{y}, \overline{q}\})$ is the *approximation state*. Let $\epsilon, \tau \in \mathbb{N}$; intuitively τ represents the "target" (or "true") function output, and ϵ represents the allowed approximation error. We say that a configuration $\mathbf{o} \in \mathbb{N}^\Lambda$ is ϵ-τ-*correct* if $|\mathbf{o}(\overline{y}) - \tau| \leq \epsilon$.

Let $f : \mathbb{N}^k \to \mathbb{N}$, $\epsilon \in \mathbb{N}$, $\mathbf{i} \in \mathbb{N}^\Lambda$, and let $\mathbf{m} = \mathbf{i} \upharpoonright \Sigma$. We say that $\mathcal{A}$ *stably* ϵ-*approximates* f *from* $\mathbf{i}$ if, for all $\mathbf{c} \in \mathsf{post}(\mathbf{i})$, there exists a $\mathbf{o} \in \mathsf{post}(\mathbf{c})$ that is stable and ϵ-$f(\mathbf{m})$-correct, i.e., from the initial configuration $\mathbf{i}$, $\mathcal{A}$ gets the output to stabilize to a value at most ϵ from the correct output. Let $S^{\mathcal{A}}_{f,\mathbf{m},\epsilon} = \{\mathbf{o} \in \mathbb{N}^\Lambda \mid \mathbf{o}$ is stable and ϵ-$f(\mathbf{m})$-correct $\}$. Note that $\mathcal{A}$ stably ϵ-approximates f from $\mathbf{i}$ if and only if $\mathsf{Pr}\left[\mathbf{i} \Longrightarrow S^{\mathcal{A}}_{f,\mathbf{m},\epsilon}\right] = 1$.

Let $\mathcal{E} : \mathbb{N} \to \mathbb{N}$; the choice of $\mathcal{E}$ as a function instead of a constant reflects the idea that the approximation error is allowed to depend on the initial count $\mathbf{i}(\overline{a})$ of the approximation state $\overline{a}$, i.e., $\mathcal{E}(\mathbf{i}(\overline{a}))$ is the desired approximation error. We say that $\mathcal{A}$ *stably* $\mathcal{E}$-*approximates* f if there are $a_0 \in \mathbb{N}$ and linearly bounded $q_0 : \mathbb{N}^{k+1} \to \mathbb{N}$ such that, for any $\mathbf{i} \in \mathbb{N}^\Lambda$, defining $\mathbf{m} = \mathbf{i} \upharpoonright \Sigma$, if $\mathbf{i}(\overline{a}) \geq a_0$, $\mathbf{i}(\overline{q}) \geq q_0(\mathbf{m}, \mathbf{i}(\overline{a}))$, and $\mathbf{i}(\overline{s}) = 0$ for all $\overline{s} \in \Lambda \setminus (\Sigma \cup \{\overline{q}, \overline{a}\})$, then $\mathcal{A}$ stably $\mathcal{E}(\mathbf{i}(\overline{a}))$-approximates f from $\mathbf{i}$.[3] An initial configuration $\mathbf{i}$ so defined is *valid*.

As we consider leaderless population protocols, we need to make sure that $\overline{a}$ does not act as a small count "leader". Consistent with the rest of this paper, we reason about initial configurations with $\mathbf{i}(\overline{a}) \geq \alpha n$ for some $\alpha > 0$ to ensure α-density.

Let $f : \mathbb{N}^k \to \mathbb{N}$. In defining running time for function-approximating population protocols, we express the expected time as a function of *both* the total number of agents

[3] I.e., the initial count $\mathbf{i}(\overline{a})$ can influence the initial required count $\mathbf{i}(\overline{q})$, since adding more initial $\overline{a}$ may imply that more quiescent agents are required as "fuel". However, a_0 is constant, not a function of $\mathbf{m}$.

$n = \|\mathbf{i}\|$ *and* the initial count $\mathbf{i}(\overline{a})$ of approximation states. Let $\mathcal{E} : \mathbb{N} \to \mathbb{N}$ and $t : \mathbb{N}^2 \to \mathbb{N}$. Given a function-approximating population protocol $\mathcal{A}$ that $\mathcal{E}$-approximates f, we say $\mathcal{A}$ $\mathcal{E}$-approximates f *in expected time* t if, for all valid initial configurations $\mathbf{i}$ of $\mathcal{A}$, letting $\mathbf{m} = \mathbf{i} \upharpoonright \Sigma$, $\mathsf{T}\left[\mathbf{i} \Longrightarrow S^{\mathcal{A}}_{f,\mathbf{m},\mathcal{E}(\mathbf{i}(\overline{a}))}\right] \leq t(n, \mathbf{i}(\overline{a}))$.

The following theorem states that given any linear function f and any population protocol $\mathcal{P}$, if f has a non-integer or negative coefficient, then $\mathcal{P}$ requires at least linear time to approximate f with sublinear error. It states this by contrapositive: if the protocol takes sublinear time, then the error $\mathcal{E} : \mathbb{N} \to \mathbb{N}$ must grow at least linearly with the initial count of approximation state $\overline{a}$. In particular, the initial configurations $\mathbf{i}$ (letting $n = \|\mathbf{i}\|$) on which our argument maximizes the error have $\mathbf{i}(\overline{a}) = \Omega(n)$. Thus, the fact that $\mathcal{E}(a) \geq \gamma a$ implies that on these $\mathbf{i}$, the error is $\Omega(n)$.

▶ **Theorem 4.1.** *Let $f : \mathbb{N}^k \to \mathbb{N}$ be a linear function that is not $\mathbb{N}$-linear. Let $\mathcal{E} : \mathbb{N} \to \mathbb{N}$. Let $\mathcal{A}$ be a function-approximating population protocol that stably $\mathcal{E}$-approximates f in expected time t, where for some $\alpha > 0$, $t(n, \alpha n) = o(n)$. Then there is a constant $\gamma > 0$ such that, for infinitely many $a \in \mathbb{N}$, $\mathcal{E}(a) \geq \gamma a$.*

A protocol stably computing f also stably $\mathcal{E}$-approximates f for $\mathcal{E}(a) = 0$, so we have:

▶ **Corollary 4.2.** *Let $f : \mathbb{N}^k \to \mathbb{N}$ be a linear function $f(\mathbf{m}) = \sum_{i=1}^{k} \lfloor c_i \mathbf{m}(i) \rfloor$, where $c_i \notin \mathbb{N}$ for some $i \in \{1, \ldots, k\}$. Let $\mathcal{C}$ be a function-computing population protocol that stably computes f. Then $\mathcal{C}$ takes expected time $\Omega(n)$.*

This gives a complete classification of the asymptotic efficiency of computing linear functions $f(\mathbf{m}) = \sum_{i=1}^{k} \lfloor c_i \mathbf{m}(i) \rfloor$ with population protocols. If $c_i \in \mathbb{N}$ for all $i \in \{1, \ldots, k\}$, then f is stably computable in logarithmic time by Observation 5.1. Otherwise, f requires linear time to stably compute by Corollary 4.2.

5 Logarithmic-time, linear-error approximation of linear functions with nonnegative rational coefficients is possible

It is easy to see that any $\mathbb{N}$-linear function f can be stably computed in logarithmic time. Recall that $\overline{x}, \overline{q} \to \overline{y}, \overline{y}$ stably computes $f(m) = 2m$ in expected time $O(\log n)$. The extension to larger coefficients and multiple inputs is routine:

▶ **Observation 5.1.** *Let $f : \mathbb{N}^k \to \mathbb{N}$ be an $\mathbb{N}$-linear function. There is a function-computing population protocol that stably computes f in expected time $O(\log n)$.*

We now describe how to stably approximate linear functions with nonnegative *rational* coefficients, i.e., $\mathbb{Q}_{\geq 0}$-linear functions, with a linear approximation error, in logarithmic time. (It is open to do this for negative coefficients, e.g., $f(m_1, m_2) = m_1 - m_2$). Recall the following simple example of a population protocol that approximately divides by 2 (that is, with probability 1 it outputs a value guaranteed to be a certain distance to the correct output), with a linear approximation error, and is fast ($O(\log n)$ time) with initial counts $\#\overline{x} = m$, $\#\overline{a} = \gamma m$, and $\#\overline{q} = \#\overline{y} = 0$:

$$\overline{a}, \overline{x} \to \overline{b}, \overline{y}$$
$$\overline{b}, \overline{x} \to \overline{a}, \overline{q}$$

which stabilizes $\#\overline{y}$ to somewhere in the interval $\{m/2, m/2 + 1, \ldots, m/2 + \gamma m\}$.

To see that the protocol is correct, note that the transition sequence can make $\#\overline{y}$ closer to one endpoint of the interval or the other depending on which transitions are chosen to consume the *last* γm of $\overline{x}$, but no matter what, the first transition executes at least as many times as the second, but not more than γm times more.

If $\#\overline{a} = 1$ initially, the above protocol stably computes $\lfloor m/2 \rfloor$ (taking linear time just for the last transition; and in total takes $\Theta(n \log n)$ time, by a coupon collector argument).

To see that the protocol takes $O(\log n)$ time if $\#\overline{a} = \gamma m$ initially, note $n = m + \gamma m \leq 2m$. Observe that $\#\overline{a} + \#\overline{b} = \gamma m$ in any reachable configuration. Thus the probability any given interaction is one of the above two transitions is $\approx \frac{\gamma m \#\overline{x}}{n^2}$, so the expected number of interactions until such a transition occurs is $\frac{n^2}{\gamma m \#\overline{x}}$. After m such transitions occur, all the input $\overline{x}$ is gone and the protocol stabilizes, which by linearity of expectation takes expected number of interactions

$$\sum_{\#\overline{x}=1}^{m} \frac{n^2}{\gamma m \#\overline{x}} = \frac{n^2}{\gamma m} \sum_{\#\overline{x}=1}^{m} \frac{1}{\#\overline{x}} \approx \frac{n^2}{\gamma m} \ln m \leq \frac{n^2}{\gamma n/2} \ln n = \frac{2n}{\gamma} \ln n,$$

i.e., expected parallel time $\frac{2}{\gamma} \ln n$. Thus this shows a tradeoff between accuracy and speed in a single protocol, adjustable by the initial count of $\overline{a}$. In this case, the approximation error increases, and the expected time to stabilization decreases, with increasing initial $\#\overline{a}$.

More generally, we can prove the following. In particular, if $a = \Omega(n)$, then $t(n, a) = O(\log n)$. Also, if $a = o(n)$, then the approximation error is $o(n)$, and if $a = \omega(\log n)$, then the expected time is $o(n)$ also. This does not contradict Theorem 4.1 since setting $a = o(n)$ implies the initial configurations are not all α-dense for a fixed $\alpha > 0$.

▶ **Theorem 5.2.** *Let $f : \mathbb{N}^k \to \mathbb{N}$ be a $\mathbb{Q}_{\geq 0}$-linear function. Let $\mathcal{E} : \mathbb{N} \to \mathbb{N}$ be the identity function. Define $t : \mathbb{N}^2 \to \mathbb{N}$ by $t(n, a) = \frac{n}{a} \log n$. Then there is a function-approximating population protocol $\mathcal{A}$ that $\mathcal{E}$-approximates f in expected time $O(t)$.*

The basic analysis is similar to the example protocol above, and the extension to rational coefficients other than $\frac{1}{2}$ follows techniques used in similar papers on function computation with chemical reaction networks [11, 13].

6 Predicate computation

In this section we show that a wide class of Boolean predicates cannot be stably computed in sublinear time by population protocols (without a leader). Intuitively, this is the class of predicates $\phi : \mathbb{N}^k \to \{0, 1\}$ such that for all $m \in \mathbb{N}$, there are two inputs $\mathbf{m}_0, \mathbf{m}_1 \in \mathbb{N}^k_{\geq m}$ such that $\phi(\mathbf{m}_0) \neq \phi(\mathbf{m}_1)$. (See the definition of *eventually constant* below.)

Formally, a *predicate-deciding population protocol* is a tuple $\mathcal{D} = (\Lambda, \delta, \Sigma, \Upsilon_1)$, where (Λ, δ) is a population protocol, $\Sigma \subseteq \Lambda$ is the set of *input states*, and $\Upsilon_1 \subseteq \Lambda$ is the set of *1-voters*. By convention, we define $\Upsilon_0 = \Lambda \setminus \Upsilon_1$ to be the set of *0-voters*. The *output* $\Phi(\mathbf{c})$ of a configuration $\mathbf{c} \in \mathbb{N}^\Lambda$ is $b \in \{0, 1\}$ if $\mathbf{c}(\overline{s}) = 0$ for all $\overline{s} \in \Upsilon_{1-b}$ (i.e., if the vote is unanimously b); the output is undefined if voters of both types are present. We say $\mathbf{o} \in \mathbb{N}^\Lambda$ is *stable* if $\Phi(\mathbf{o})$ is defined and for all $\mathbf{o}' \in \mathsf{post}(\mathbf{o})$, $\Phi(\mathbf{o}') = \Phi(\mathbf{o})$. For all $\mathbf{m} \in \mathbb{N}^k$, define initial configuration $\mathbf{i}_\mathbf{m} \in \mathbb{N}^\Lambda$ by $\mathbf{i}_\mathbf{m} \upharpoonright \Sigma = \mathbf{m}$ and $\mathbf{i}_\mathbf{m} \upharpoonright (\Lambda \setminus \Sigma) = \mathbf{0}$. Call such an initial configuration *valid*. For any valid initial configuration $\mathbf{i}_\mathbf{m} \in \mathbb{N}^\Lambda$ and predicate $\phi : \mathbb{N}^k \to \{0, 1\}$, let $S_{\mathbf{i}_\mathbf{m}, \phi} = \{\mathbf{o} \in \mathbb{N}^\Lambda \mid \mathbf{i}_\mathbf{m} \Longrightarrow \mathbf{o}, \mathbf{o} \text{ is stable, and } \Phi(\mathbf{o}) = \phi(\mathbf{m})\}$. A population protocol *stably decides* a predicate $\phi : \mathbb{N}^k \to \{0, 1\}$ if, for any valid initial configuration $\mathbf{i}_\mathbf{m} \in \mathbb{N}^\Lambda$, $\mathsf{Pr}[\mathbf{i}_\mathbf{m} \Longrightarrow S_{\mathbf{i}_\mathbf{m}, \phi}] = 1$. This is equivalent to requiring that for all $\mathbf{c} \in \mathsf{post}(\mathbf{i}_\mathbf{m})$, there is $\mathbf{o} \in \mathsf{post}(\mathbf{c})$ such that $\mathbf{o}$ is stable and $\Phi(\mathbf{o}) = \phi(\mathbf{m})$.

For example, the protocol defined by transitions

$$\overline{x}_1, \overline{x}_2 \to \overline{q}_1, \overline{q}_2$$
$$\overline{x}_1, \overline{q}_2 \to \overline{x}_1, \overline{q}_1$$
$$\overline{x}_2, \overline{q}_1 \to \overline{x}_2, \overline{q}_2$$
$$\overline{q}_1, \overline{q}_2 \to \overline{q}_1, \overline{q}_1$$

if $\Upsilon_1 = \{\overline{x}_1, \overline{q}_1\}$ and $\Upsilon_0 = \{\overline{x}_2, \overline{q}_2\}$, decides whether $m_1 = \mathbf{i}(\overline{x}_1) \geq m_2 = \mathbf{i}(\overline{x}_2)$. The first transition stops once the less numerous input state is gone. If $\overline{x}_1$ (resp. $\overline{x}_2$) is left over, then the second (resp. third) transition converts $\overline{q}_i$ states to its vote. If neither is left over (i.e., if $m_1 = m_2$, requiring output 1), the fourth transition converts all $\overline{q}_2$ states to $\overline{q}_1$.

Let $\phi : \mathbb{N}^k \to \{0,1\}$, and for $b \in \{0,1\}$, define $\phi^{-1}(b) = \{\mathbf{m} \in \mathbb{N}^k \mid \phi(\mathbf{m}) = b\}$ to be the set of inputs on which ϕ outputs b. We say ϕ is *eventually constant* if there is $m_0 \in \mathbb{N}$ such that ϕ is constant on $\mathbb{N}^k_{\geq m_0} = \{\mathbf{m} \in \mathbb{N}^k \mid (\forall i \in \{1, \ldots, k\})\ \mathbf{m}(i) \geq m_0\}$, i.e., either $\phi^{-1}(0) \cap \mathbb{N}^k_{\geq m_0} = \emptyset$ or $\phi^{-1}(1) \cap \mathbb{N}^k_{\geq m_0} = \emptyset$. In other words, although ϕ may have an infinite number of each output, "sufficiently far from the boundary" (where all coordinates exceed m_0), only one output appears.

The following theorem shows that any predicate that is not eventually constant cannot be stably decided in sublinear time by a population protocol.

▶ **Theorem 6.1.** *Let $\phi : \mathbb{N}^k \to \{0,1\}$ and $\mathcal{D}$ be a predicate-deciding population protocol that stably decides ϕ. If ϕ is not eventually constant, then $\mathcal{D}$ takes expected time $\Omega(n)$.*

Alistarh, Aspnes, Eisenstat, Gelashvili, and Rivest [1] showed a linear-time lower bound on any leaderless population protocol deciding the majority predicate. Recall that their technique is based on showing that after adding enough of the input in the minority to change it to the majority, the effect of this addition can be effectively nullified by surgery of the transition sequence, yielding a stable configuration with the original (now incorrect) answer. The technique can be extended easily to show various other specific predicates, such as equality and parity, also require linear time. We use the same technique of finding pairs of inputs with opposite correct answers and apply a similar transition sequence surgery. The main difficulty in showing Theorem 6.1, which covers the class of *all* predicates that are semilinear but not eventually constant, is to identify a common characteristic that can be exploited to find pairs of inputs that are α-dense for some $\alpha > 0$. Here, we rely on the semilinear structure of the predicate computed. Indeed, note that we cannot find such α-dense pairs for the predicate $\phi : \mathbb{N}^2 \to \{0,1\}$ with support $\{(k, 2^k) \mid k \in \mathbb{N}\}$, which is not eventually constant (but also not semilinear).

7 Open Questions

Time complexity of other functions. What is the optimal time complexity of computing semilinear functions and predicates not satisfying the hypotheses of Theorems 3.1 and 6.1; namely the *eventually-$\mathbb{N}$-linear* functions, (e.g., $f(m) = 0$ if $m < 3$ and $f(m) = m$ otherwise) and *eventually-constant* predicates (e.g., $\phi(m) = 1$ iff $m \geq 2$)?

Stabilization vs convergence. Measuring time to stabilization in the randomized model, as we do here, measures the expected time until the probability of changing the output becomes 0. Our proof shows only that stabilization must take expected $\Omega(n)$ time. However, convergence could occur much earlier in a transition sequence than stabilization (we can say

a particular transition sequence *converged* at the point when the output count is the same in every subsequently reached configuration). We conjecture that similar negative results hold for convergence for leaderless population protocols. It is also open whether stabilization can occur in sublinear time, *even with an initial leader*. The known stably computing protocols converging in $O(\log^5 n)$ time [5, 11] provably require expected time $\Omega(n)$ to stabilize.

Acknowledgements. We are grateful to Sungjin Im for the proof of an important technical lemma, and we thank anonymous reviewers for very helpful comments.

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in molecular computation. In *SODA 2017: Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2017. to appear.

2 Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *ICALP 2015: Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming, Kyoto, Japan*, 2015.

3 Dana Angluin, James Aspnes, Zoë Diamadi, Michael Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18:235–253, 2006. Preliminary version appeared in PODC 2004. doi:10.1007/s00446-005-0138-3.

4 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *PODC 2006: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 292–299, New York, NY, USA, 2006. ACM Press. doi:10.1145/1146381.1146425.

5 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, September 2008. Preliminary version appeared in DISC 2006.

6 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

7 Alexandre Baccouche, Kevin Montagne, Adrien Padirac, Teruo Fujii, and Yannick Rondelez. Dynamic dna-toolbox reaction circuits: a walkthrough. *Methods*, 67(2):234–249, 2014.

8 James M Bower and Hamid Bolouri. *Computational modeling of genetic and biochemical networks*. MIT press, 2004.

9 E. Cardoza, Richard J. Lipton, and Albert R. Meyer. Exponential space complete problems for Petri nets and commutative semigroups (preliminary report). In *STOC 1976: Proceedings of the 8th annual ACM Symposium on Theory of Computing*, pages 50–54. ACM, 1976.

10 Ho-Lin Chen, Rachel Cummings, David Doty, and David Soloveichik. Speed faults in computation by chemical reaction networks. *Distributed Computing*, 2015. to appear. Special issue of invited papers from DISC 2014.

11 Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Natural Computing*, 13(4):517–534, 2014. Preliminary version appeared in DNA 2012.

12 Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from DNA. *Nature Nanotechnology*, 8(10):755–762, 2013.

13 David Doty and Monir Hajiaghayi. Leaderless deterministic chemical reaction networks. *Natural Computing*, 14(2):213–223, 2015. Preliminary version appeared in DNA 2013.

14 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 2016. to appear. Special issue of invited papers from DISC 2015.

15 Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966. URL: `http://projecteuclid.org/euclid.pjm/1102994974`.

16 Richard M Karp and Raymond E Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

17 Ernst W Mayr and Albert R Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in mathematics*, 46(3):305–329, 1982.

18 Othon Michail and Paul G Spirakis. How many cooks spoil the soup? In *International Colloquium on Structural Information and Communication Complexity*, pages 3–18. Springer, 2016.

19 Carl A Petri. Communication with automata. Technical report, DTIC Document, 1966.

20 Niranjan Srinivas. *Programming chemical kinetics: Engineering dynamic reaction networks with DNA strand displacement*. PhD thesis, California Institute of Technology, 2015.

21 Chris Thachuk, Erik Winfree, and David Soloveichik. Leakless DNA strand displacement systems. In *DNA 2015: Proceedings of the 21st International Conference on DNA Computing and Molecular Programming*, pages 133–153. Springer, 2015.

22 Vito Volterra. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Mem. Acad. Lincei Roma*, 2:31–113, 1926.

23 David Yu Zhang and Georg Seelig. Dynamic dna nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103–113, 2011.