Yale University
Department of Computer Science

P.O. Box 208205
New Haven, CT 06520–8285

Computation in Networks of Passively Mobile Finite-State Sensors

Dana Angluin[1] James Aspnes[1,2] Zoë Diamadi[1,3]
Michael J. Fischer[1,4] René Peralta[1,4]

YALEU/DCS/TR-1281
February 23, 2004

[1]Yale University Department of Computer Science.

[2]Supported in part by NSF grants CCR-9820888, CCR-0098078, and CCR-0305258.
[3]Supported in part by ONR grant N00014-01-1-0795.
[4]Supported in part by NSF grant CSE-0081823.

Computation in Networks of Passively Mobile Finite-State Sensors

Dana Angluin* James Aspnes*† Zoë Diamadi*‡ Michael J. Fischer*§
René Peralta*§

1 Scenario: A flock of birds

Suppose we have equipped each bird in a particular flock with a sensor that can determine whether the bird's temperature is elevated or not, and we wish to know whether at least 5 birds in the flock have elevated temperatures. We assume that the sensors are quite limited: each sensor has a constant number of bits of memory and can respond to a global start signal, and two sensors can communicate only when they are sufficiently close to each other.

In this scenario, the sensors are mobile, but have no control over how they move, that is, they are **passively mobile**. Initially, we assume that the underlying pattern of movement guarantees a fairness condition on the interactions: every pair of birds in the flock repeatedly come sufficiently close to each other for their sensors to communicate.

Under these assumptions, there is a simple protocol ensuring that every sensor eventually contains the correct answer. At the global start signal, each sensor makes a measurement, resulting in a 1 (elevated temperature) or 0 (not elevated temperature) in a counter that can hold values from 0 to 4. When two sensors communicate, one of them sets its counter to the sum of the two counters, and the other one sets its counter to 0. If two counters ever sum to at least 5, the sensors go into a special alert state, which is then copied by every sensor that encounters it. The output of a sensor is 0 if it is not in the alert state, and 1 if it is in the alert state. If we wait a sufficient interval after we issue the global reset, we can retrieve the correct answer from any of the sensors.

Now consider the question of whether at least 5% of the birds in the flock have elevated temperatures. Is there a protocol to answer this question in the same sense, without assumptions about the size of the flock? In Section 3 we show that such a protocol exists. More generally, we are interested in fundamental questions about the computational power of this and related models of interactions among members of a distributed population of finite-state agents.

2 A wider view

Most work in distributed algorithms assumes that the agents in a system are computationally powerful, capable of storing non-trivial amounts of data and carrying out complex calculations. But in systems consisting of massive amounts of cheap, bulk-produced hardware, or of small mobile

*Yale University Department of Computer Science.
†Supported in part by NSF grants CCR-9820888, CCR-0098078, and CCR-0305258.
‡Supported in part by ONR grant N00014-01-1-0795.
§Supported in part by NSF grant CSE-0081823.

agents that are tightly constrained by the systems they run on, the resources available at each agent may be severely limited. Such limitations are not crippling if the system designer has fine control over the interactions between agents; even finite-state agents can be regimented into cellular automata with computational power equivalent to Turing machines. But if the system designer cannot control these interactions, it is not clear what the computational limits are.

Sensor networks are a prime example of this phenomenon. Each sensing unit is a self-contained physical package including its own power supply, processor and memory, wireless communication capability, and one or more sensors capable of registering information about the local environment of the unit. Constraints on cost and size translate into severe limitations on power, storage, processing, and communication. Sensing units are designed to be deployed in large groups, using local low-power wireless communication between units to transmit information from the sensors back to a base station or central monitoring site.

Research in sensor networks has begun to explore the possibilities for using distributed computation capabilities of networks of sensors in novel ways to reduce communication costs. Aggregation operations, such as count, sum, average, extrema, median, or histogram, may be performed on the sensor data in the network as it is being relayed to the base station [9, 10]. Flexible groups of sensors associated with targets in spatial target tracking can conserve resources in inactive portions of the tracking area [7, 14]. Though sensors are usually assumed to be stationary or nearly so, permitting strategies based on relatively stable routing, this assumption is not universal in the sensor-network literature. For example, an assumption of random mobility and packet relay dramatically increases the throughput possible for communication between source-destination pairs in a wireless network [8].

The flock of birds scenario illustrates the question of characterizing what computations are possible in a cooperative network of passively mobile finite-state sensors. The assumptions we make about the motion of the sensors are that it is passive (not under the control of the sensors), that it is sufficiently rapid and unpredictable for stable routing strategies to be infeasible, and that each pair of sensors will repeatedly be close enough to communicate using a low power wireless signal.

There is a global start signal transmitted by the base station to all the sensors simultaneously to initiate a computation. When they receive the global start signal, the sensors take a reading (one of a finite number of possible input values) and attempt to compute some function or predicate of all the sensor values. This provides a "snapshot" of the sensor values, rather than the continuous stream of sensor values more commonly considered. Sensors communicate in pairs, and do not have unique identifiers; thus, they update their states based strictly on the pair of their current states. We assume that they are capable of symmetry-breaking, allowing them to go from two identical states to two different states.

In Section 3, we define a model of computation by pairwise interactions in a population of identical finite-state agents. Assuming a fairness condition on interactions, we define the concept of eventual computation of a function or predicate by a population protocol. We give protocols for threshold k, parity, majority, and simple arithmetic functions, as well as closure results that allow us to define a useful expression language capturing a subset of the power of this model. We also show that every predicate computable in this model is in nondeterministic log space. An open problem is to give an exact characterization of the computational power of eventual computation.

In Section 4 we add a uniform sampling condition on interactions, to obtain the model of conjugating automata. This allows us to consider computations that are correct with high probability, and to address questions of expected resource use. We show that this model has sufficient power to simulate, with high probability, a counter machine with $O(1)$ counters of capacity $O(n)$. We further show that Boolean predicates computable with high probability in this model are in $P \cap RL$. This gives a partial characterization of the set of predicates computable by such machines, but finding an exact characterization is still open. In Section 5 we describe other related work, and in Section 6 we discuss some of the many intriguing questions raised by these models.

3 What can be computed eventually?

We define a model generalizing the flock of birds scenario from Section 1 and examine its computational power and limitations.

3.1 A model of pairwise interaction

A **population protocol** $\mathcal{A}$ consists of finite input and output alphabets X and Y, a finite set of states Q, a function $I : X \to Q$ mapping inputs to states, a function $O : Q \to Y$ mapping states to outputs, and a transition function $\delta : Q \times Q \to Q \times Q$, such that $\delta(q_1, q_2) = \delta(q_2, q_1)$ for all $q_1, q_2 \in Q$.

As a simple illustration, we formalize the count-to-five protocol from Section 1. The states are q_i for $0 \leq i \leq 5$. The input and output alphabets are $X = Y = \{0, 1\}$. The input function I maps 0 to q_0 and 1 to q_1. The output function O maps all states except q_5 to 0 and the state q_5 to 1. The transition function $\delta(q_i, q_j)$ is defined as follows: if $i + j \geq 5$, then the result is (q_5, q_5), otherwise, the result is (q_{i+j}, q_0).

A **population configuration** consists of a multiset of elements of Q specifying the state of each member of the population. Let C_1 and C_2 be two population configurations. We say that C_2 is a **successor of** C_1, denoted $C_1 \to C_2$, if there exist states q_1 and q_2 such that C_2 can be obtained from C_1 by applying the transition rule to the pair (q_1, q_2), that is, $\{q_1, q_2\}$ is a submultiset of C_1, and $C_2 = (C_1 - \{q_1, q_2\}) + \delta(q_1, q_2)$, where $-$, $+$, and $=$ denote multiset difference, union and equality, respectively.[1]

An **input** to a population protocol is a multiset x of input symbols, and the corresponding **input configuration** is $I(x)$, the multiset of corresponding states. The **output** of a population configuration C is $O(C)$, the multiset of output symbols corresponding to the states in C.

A **computation** is a finite or infinite sequence of population configurations $C_0, C_1, C_2, \ldots$ such that for each i, $C_i \to C_{i+1}$. An infinite computation is **pairwise-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 the consecutive pair C followed by C' also occurs infinitely often in the computation. The **output** of a finite computation is the output of its last configuration. The output of an infinite computation **stabilizes** if there is an index i such that for all $j \geq i$, $O(C_j) = O(C_i)$, in which case we say that $O(C_i)$ is the **output** of the infinite computation.

Continuing our count-to-five illustration, the input $x = \{0, 0, 1, 1, 1, 1\}$ is mapped to the input configuration:

$$I(x) = \{q_0, q_0, q_1, q_1, q_1, q_1\},$$

[1] Formally, $\delta(q_1, q_2)$ is a tuple. We often identify tuples and multisets when the ordering of elements is immaterial.

from which the following is a possible computation:

$$\{q_0, q_0, q_1, q_1, q_1, q_1\} \to \{q_0, q_0, q_0, q_1, q_1, q_2\} \to \{q_0, q_0, q_0, q_0, q_2, q_2\} \to \{q_0, q_0, q_0, q_0, q_0, q_4\}.$$

The last configuration repeats indefinitely; its output is $\{0, 0, 0, 0, 0, 0\}$.

Let f be a function from multisets over X to multisets over Y. We say that a population protocol $\mathcal{A}$ **eventually computes** f if for every input x and every pairwise-fair computation of $\mathcal{A}$ starting with the corresponding input configuration $I(x)$, the output stabilizes to $f(x)$.

For a function g mapping X^n to Y (which includes predicates when $Y = \{0, 1\}$) we say that $\mathcal{A}$ eventually computes g if it eventually computes h, where the value of $h(x)$ is the multiset consisting of n copies of $g(x)$. That is, we require that every pairwise-fair computation eventually reach only configurations in which every output is $g(x)$. Clearly such a function must be symmetric, since the order of inputs is not preserved.

3.2 What functions are eventually computable?

We start by considering families of Boolean functions $\{f_n\}$ such that $f_n : \{0, 1\}^n \to \{0, 1\}$ for all $n \geq 1$.

Closure properties. Let $\{f_n\}$ and $\{g_n\}$ be families of Boolean functions eventually computable by population protocols $\mathcal{A}$ and $\mathcal{B}$. By complementing the output map of $\mathcal{A}$, the family of negations $\{f'_n\}$ is eventually computable. By complementing the input map of $\mathcal{A}$, the family of functions $\{h_n\}$, where h_n is equal to f_n with its inputs complemented, is eventually computable. Product constructions with $\mathcal{A}$ and $\mathcal{B}$ yield population protocols that eventually compute the conjunction or disjunction of the outputs of $\{f_n\}$ and $\{g_n\}$. Formally, we have:

Lemma 1 *Let $\{f_n\}$ and $\{g_n\}$ be families of Boolean functions. If $\{f_n\}$ and $\{g_n\}$ are eventually computable, then so are $\{\neg f_n\}$, $\{f_n \wedge g_n\}$, and $\{f_n \vee g_n\}$.*

Thus, in addition to predicates such as "at least 5 ones" we can compute predicates such as "fewer than 5 ones", "at least 5 zeros", "at most 5 zeros", "exactly 5 ones", "exactly 5, 7 or 9 ones" and so on. Though it is not immediately obvious, parity and majority are also eventually computable by population protocols.

Parity. Our construction for parity uses a state consisting of two bits: the data bit and the live bit. Initially, the data bit is equal to the input bit, and the live bit is 1. For each state, the output bit is equal to the data bit. When two members meet whose live bits are both 1, one sets its live bit to 0, and the other sets its data bit to the mod 2 sum of their data bits. When a member with live bit 0 meets a member with live bit 1, the former copies the data bit of the latter.

In this protocol, the mod 2 sum of the product of the live bit and the data bit over all the members of the population is invariant and equal to the mod 2 sum of the inputs. Eventually, exactly one member of the population has its live bit set to 1. At that point, its data bit is the correct output. Once there is a single live bit set to 1, eventually every other member copies the (correct) data bit from that member.

The live bit ensures a tree-structured computation aggregating the data bits, with the final result at the root of the tree. This generalizes to the computation of the product of all the input values in a commutative semigroup. Thus, all the symmetric regular languages are eventually computable, e.g., deciding whether the number of 1's is i modulo m for constants i and m.

Majority. The value of the majority function is 1 if there are more 1's than 0's in the input, otherwise it is 0, a symmetric, non-regular predicate.

The states of our protocol consist of a live bit and a counter with values in the set $\{-1, 0, 1\}$. Initially, the live bit is 1, and the counter is -1 if the input is 0 and 1 if the input is 1. The output is 1 if the counter is 1, otherwise it is 0. When two members with live bit equal to 1 meet, if the sum of their counters is in the set $\{-1, 0, 1\}$, then both set their counters to the sum, and one of the members sets its live bit to 0; otherwise, they do nothing. When a member with live bit equal to 0 meets a member with live bit equal to 1, the former copies the counter value of the latter.

In this protocol, the sum of the counters for all population members with live bit equal to 1 is invariant and equal to the number of 1's minus the number of 0's in the input. Eventually there will remain one or more live bits equal to 1, with no more combinations possible, which means that the associated counter(s) have a single value -1, 0, or 1, indicating that the number of 1's in the input is less than, equal to, or greater than the number of 0's in the input, respectively. After this point, every other population member will copy the common counter value, and their outputs will also be correct.

A generalization with counters capable of holding integer values between $-k$ and k inclusive determines whether at least a fraction $1/(k+1)$ of the inputs are 1's, demonstrating the existence of a population protocol to detect whether at least 5% of the flock of birds have elevated temperatures, as claimed in Section 1.

Arithmetic functions In addition to computing Boolean functions, population protocols can compute many simple arithmetic functions. We can represent $O(1)$ integers with values up to $O(n)$ in a population configuration of size n by using a composite state that contains a constant-bounded counter for each integer being represented. For example, the state $(-3, 0, 5)$ could indicate negative three units of the first integer, zero units of the second integer, and five units of the third integer. The value represented for each integer is obtained by summing the corresponding components over the whole population.

Using this representation, there are population protocols to compute the sum of two integers, the product of an integer and a constant, the integer quotient of an integer and a constant, and the value of an integer modulo a constant. These protocols produce outputs that are eventually stable provided their inputs are eventually stable, which allows them to be composed freely.

An eventually computable expression language. Putting together the base functions and closure results, we can define an expression language such that the expressible predicates are eventually computable by population protocols. This gives a lower bound on the set of eventually computable predicates: whether every eventually computable predicate is so expressible is an open problem.

For each input symbol $\sigma \in X$, there is a variable N_σ representing the number of copies of σ in the input multiset. Constants are nonnegative integers. Each term is a constant, a variable, or the sum of two terms, or the product of a constant and a term, or the (integer) quotient of a term and a nonzero constant, or the remainder of a term modulo a nonzero constant. Atomic expressions are formed from two terms and one of the predicates: $=, \leq, <, \geq, >$. An expression is either an atomic expression or the negation of an expression, the conjunction of two expressions, or the disjunction of two expressions. Each expression is either true or false of a given multiset of inputs, by the usual semantics.

For example, the count-to-five problem is expressible as $(N_1 \geq 5)$, the parity problem as

$((N_1 \bmod 2) = 1)$, and the majority problem as $(N_1 > N_0)$. The question of whether the number of 1's is between 15% and 20% of the total population can be expressed as

$$((17N_1 \geq 3N_0) \wedge (4N_1 \leq N_0)).$$

This idea extends to non-binary input alphabets, so that if $X = \{a, b, c\}$, we can express the predicate that the number of a's, b's, and c's are equal by the expression $(N_a = N_b) \wedge (N_b = N_c)$.

Theorem 2 *Any predicate expressed in the language described above is eventually computable.*

3.3 What functions are not eventually computable?

Theorem 2 gives a partial characterization of the eventually-computable predicates in the population model. We do not know if this characterization is complete. However, we can obtain an upper bound on the set of eventually computable predicates by showing that it is contained in the complexity class NL.

Each configuration of a population of n members can be represented by $|Q|$ counters of $O(\log n)$ bits each. Given a population protocol $\mathcal{A}$ that computes the characteristic function of a symmetric language L over the alphabet X, there is a nondeterministic Turing machine to accept L in space $O(\log n)$. To accept input x, the Turing machine must verify two conditions: that there is a configuration C reachable from $I(x)$ in which all states have output 1, and there is no configuration C' reachable from C in which some state has output 0. The first condition is verified by guessing and checking a polynomial-length sequence of population configurations reaching such a C. The second condition is the complement of a similar reachability condition. It is in nondeterministic $O(\log n)$ space because this class is closed under complement. It follows that:

Theorem 3 *All eventually computable predicates are in the class NL.*

It is an open problem to characterize exactly the power of this model of eventual computation. Concretely, we conjecture that predicates such as "the number of 1's is a power of 2" and "the number of c's is the product of the number of a's and the number of b's" are not eventually computable by population protocols. Our intuition is that the model lacks the ability to sequence or iterate computations, and we suspect that a pumping lemma of some form exists for the model.

4 Randomized interactions: conjugating automata

"Eventually" is probably not a strong enough guarantee for most practical situations, but it is the best we can offer given only the pairwise-fairness condition. To make stronger guarantees, we must put some constraints on the interactions between members of the population.

Let us add a probabilistic assumption on how the next pair to interact is chosen. Many assumptions would be reasonable to study. We consider one of the simplest: the pair to interact is chosen at random, independently and uniformly from all pairs of members of the population. This is the model of **conjugating automata**, inspired by models introduced by Diamadi and Fischer to study the acquisition and propagation of knowledge about trustworthiness in populations of interacting agents [4].

Random pairing is sufficient to guarantee pairwise fairness with probability 1, so any protocol that eventually computes a predicate g in the pairwise-fair model computes g with probability 1 on every input in the random-pairing model.

However, probabilities also allow us to consider problems where we only compute the correct answer with high probability, or to describe the expected number of interactions until a protocol converges. Given a function f mapping multisets over X to multisets over Y, a population protocol $\mathcal{A}$, and an input x, we define the probability that $\mathcal{A}$ computes f on input x to be the probability of all computations beginning with $I(x)$ that stabilize with output $f(x)$.

For example, for the parity protocol, the expected number of interactions in a computation until there is just one live bit equal to 1 is $\Theta(n^2)$, and the expected number of further interactions until every other member of the population has interacted with the unique member with live bit equal to 1 is $\Theta(n^2 \log n)$. Thus the expected total number of interactions until the output is correct is $\Theta(n^2 \log n)$. In general, we are interested in protocols that accomplish their tasks in an expected number of interactions polynomial in n, the population size.[2]

Generalizing this argument, we obtain the following by structural induction:

Theorem 4 *Let P be a predicate defined by the language of Section 3.2. Then there is a randomized population protocol that computes P with probability 1, where the population converges to the correct answer in expected total number of interactions $O(n^2 \log n)$.*

4.1 The benefits of a leader: simulating counters

If we are allowed to designate a leader in the input configuration, that is, one member of the population starts in a distinguished state, then the leader can organize the rest of the population to simulate a counter machine with $O(1)$ counters of capacity $O(n)$, succeeding with high probability. To simulate the contents of the counters, we use the representation described in Section 3 for integers in arithmetic computations, but using only nonnegative component counter values. We assume that the inputs to the counter machine are supplied in designated counters (which can be emulated by the input map I), and the leader simulates the finite-state control of the counter machine.

To decrement a particular simulated counter, the leader waits to encounter a population member with the corresponding component of its state greater than zero, and decrements it. Incrementing is similar; the relevant component must be less than its maximum. These operations will happen with probability 1, assuming that they are possible.

However, testing a counter for zero is different; the leader must attempt to decide whether there are any population members with the corresponding component greater than zero. We give a method that is correct with high probability. It is the ability to make (possibly incorrect) decisions that enables effective sequencing and iteration of computations in this model.

The leader initially labels one other member of the population (the timer) with a special mark. The leader waits for one of the two events: (1) an interaction with a population member with a nonzero in the state component corresponding to the simulated counter, or (2) k consecutive interactions with the timer. If an event of type (1) occurs first, then the simulated counter is certainly not zero. The event (2) has low probability, so if it occurs first, the probability is high that the leader has encountered every member of the population in the meantime, and may (with

[2]Note that such protocols do not terminate with a final answer; they remain capable of resuming indefinitely.

a small probability of error) conclude that the simulated counter is zero. The parameter k controls the probability of error, at the expense of increasing the expected number of interactions.

Lemma 5 *If the leader marks one member of the population as a timer, the expected total number of interactions until the leader encounters the timer k times in a row is $\Theta(n^{k+1})$.*

Rather than sum error bounds for individual counter operations, we analyze the error associated with the macro-operations on counters:

Lemma 6 *For sufficiently large k, the operations of zeroing a counter or zeroing a counter while adding its contents to one or more other counters can be performed in an expected total number of interactions $\Theta(n^{k+1})$ with $O((1/n)^{k-1} \cdot \log n)$ probability of error. The same holds for the operations of multiplying by a constant, integer quotient with a constant, or remainder modulo a constant.*

4.2 How to elect a leader

If we do not have a unique leader in the input configuration, it is possible to establish one using the ideas of the live bit, as in the parity and majority protocols of Section 3.2, and the timer mark of Section 4.1.

At the global start signal, every member of the population sets its input symbol (which it remembers for the duration of the computation), sets its live bit equal to 1, and clears its timer mark (indicating that it is not a timer). Any member of the population whose live bit equals 1 begins an initialization phase: it marks the first non-timer member of the population that it encounters as a timer and attempts to initialize every other member of the population. It uses the event of encountering a timer k times in a row to determine the end of the initialization phase.

Of course, at first every member of the population is attempting to run the initialization phase, so there will be general chaos. Whenever two members of the population with live bit equal to 1 encounter each other, one sets its live bit to 0. If that member has already marked a timer, the *other* member, which keeps its live bit equal to 1, waits until it encounters a timer and turns it back into a non-timer before proceeding. It then restarts the initialization phase (not creating another timer if it has already released one). When initialized, population members with live bit equal to 0 revert to a state representing only their input and their live bit, but they retain their timer status.

If a population member with live bit equal to 1 completes the initialization phase, it begins the computation (e.g., simulating a counter machine, as in the preceding section). If during the computation it encounters another population member with live bit equal to 1, the two proceed as indicated above, one setting its live bit to 0, and the other restarting the initialization phase, with appropriate housekeeping to ensure retrieval of the extra timer, if any.

After a period of unrest lasting an expected $\Theta(n^2)$ interactions, there will be just one population member with live bit equal to 1. After the interaction eliminating the last rival, this lucky winner will succeed in initializing all other population members with high probability (because there is only one timer in the population) and proceed with the computation as the unique leader. If and when the counter machine halts, the unique leader can propagate that fact (along with the output, if a function of one output is being computed) to all the other population members. If there have been no errors during the (final) simulation, the output of every configuration in the rest of the computation is correct.

We have just shown that we can carry out the operations of a counter machine with high probability. Using a standard construction due to Minsky [12], this allows us to simulate randomized log-space Turing machines with high probability.

Corollary 7 *Let $f(x)$ be a function in randomized log-space, where the input x is represented in unary. Then for any fixed c, there is a population protocol that, with n members, computes $f(x)$ for any $x \leq n$ with probability of error $O(n^{-c})$ in expected time polynomial in n.*

4.3 Simulating randomized population protocols

In this section, we show either deterministic polynomial time or randomized logarithmic space is sufficient to simulate the results of a computation by a population protocol with random pairing.

Suppose that a population protocol $\mathcal{A}$ computes a function f with probability at least $2/3$. Then f can be computed by a polynomial-time Turing machine. As before, we assume that a string x of symbols from X represents the multiset input x to $\mathcal{A}$, so that n represents both the input length and the population size.

A polynomial-time Turing machine can construct the matrix representing the Markov chain whose states are the population configurations reachable from $I(x)$, since there are at most $n^{|Q|}$ of them. Solving for the stationary probability of the configurations, the Turing machine can determine a set of configurations of probability at least $1/2$, each of which has the correct output, which the Turing machine then writes to its output tape.

Also under these assumptions, f can be computed by a randomized Turing machine with probability $2/3-\epsilon$ using space $O(\log n)$. A randomized Turing machine simulates the population protocol by using a finite number of $O(\log n)$-bit counters to keep track of the number of members of the population in each state. Using coin flips, it simulates drawing a random pair of population members and updating the counters according to the transition function of $\mathcal{A}$. By running the simulation for long enough, the randomized Turing machine can be almost certain of being in a terminal strongly connected component of the states of the Markov chain, at which point the Turing machine halts and writes the output of the current configuration on its output tape.

To wait sufficiently long, the randomized Turing machine allocates a counter of $O(\log n)$ bits and flips a coin before each simulated interaction, keeping track of the number of consecutive heads in the counter. The simulation is stopped when the counter overflows, which gives an expected number of simulated interactions before termination that is at least 2^{n^c}.

We have just shown:

Theorem 8 *The set of Boolean predicates accepted by a randomized population protocol is contained in $P \cap RL$.*

5 Other related work

In a Petri net, a finite collection of tokens may occupy one of a finite set of places, and transition rules specify how the tokens may move from place to place.[3] Viewing the states of a population protocol as places, and the population members as tokens, our models can also be interpreted as particular kinds of Petri nets. Randomized Petri nets were introduced by Volzer [13] using a transition rule that does not depend on the number of tokens in each input place, as ours does in the case of conjugating automata.

The Chemical Abstract Machine of Berry and Boudol [2] is an abstract machine designed to model a situation in which components move about a system and communicate when they come

[3]See [5, 6] for surveys of Petri nets.

into contact, based on a metaphor of molecules in a solution governed by reaction rules. A concept of enforced locality using membranes to confine subsolutions allows the machines to implement classical process calculi or concurrent generalizations of the lambda calculus.

Brand and Zafiropulo [3] define a model of communicating processes consisting of a collection of finite state machines that can communicate via pre-defined FIFO message queues, and focus on general properties of protocols defined in the model, such as the possibility of deadlock or loss of synchronization.

Milner's bigraphical reactive systems [11] address the issues of modeling locality and connectivity of agents by two distinct graph structures. In this work the primary focus is upon the expressiveness of the models, whereas we consider issues of computational power and resource usage.

6 Open problems

In addition to the open problem of characterizing the power of eventual computation, many other intriguing questions and directions are suggested by this work. One direction we have explored [1] is to define a novel storage device, the **urn**, which contains a multiset of tokens from a finite alphabet and functions as auxiliary storage for a finite control with input and output tapes, analogous to the pushdown or work tape of traditional models. Access to the tokens in the urn is by uniform random sampling, making it similar to the model of conjugating automata.

The models in this paper assume a "snapshot" of the inputs is taken when the global start signal is received. A model accommodating streaming inputs, as is typically assumed in sensor networks, would be very interesting.

We have assumed uniform sampling of pairs to interact, but for some applications it may make sense to consider other sampling rules. One idea is weighted sampling, in which population members are sampled according to their weights, possibly depending on their current states. We conjecture that with reasonable restrictions on the weights, weighted sampling yields the same power as uniform sampling. Other sampling rules might be based on more accurate models of patterns of interaction in populations of interest.

The interaction rules we consider are deterministic and specify pairwise interactions. What happens if the rules are nondeterministic, or specify interactions of larger groups, or allow the interaction to increase or decrease the population?

We give bounds on the expected total number of interactions, but other resource measures may be more appropriate in some applications. For many applications, interactions will happen in parallel, so that the total number of interactions may not be well correlated with wall-clock time; defining a useful notion time is a challenge. Alternatively, if we consider only the number of interactions in which at least one state changes (which might be correlated with the energy required by the computation), then the bounds can be finite even in the eventual computation model, and the expected bounds can be smaller in the conjugating automata model.

7 Acknowledgments

The authors wish to thank Richard Yang for valuable advice regarding these ideas and David Eisenstat for the parity construction.

References

[1] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Urn automata. Manuscript, November 2003.

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

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

[4] 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`.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.

[13] H. Volzer. Randomized non-sequential processes. In *Proceedings of CONCUR 2001-Concurrency Theory*, pages 184–201, Aug. 2001.

[14] 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.