Distributed Computing Column 71

Recent Algorithmic Advances in Population Protocols

Jennifer L. Welch
Department of Computer Science and Engineering
Texas A&M University, College Station, TX 77843-3112, USA
welch@cse.tamu.edu

Population protocols have been an active area of distributed computing for over a decade, as they are an abstract model of numerous real-world scenarios from sensor networks to biological computing paradigms. For this column, Dan Alistarh and Rati Gelashvili provide a timely overview of recent results on population protocols, focusing especially on the case when each agent in the system can have more than a constant amount of local memory. Several key building blocks are presented, followed by applications of these primitives, in an accessible way that conveys the excitement of fast-moving developments in the area.

Many thanks to Dan and Rati for their contribution!

Call for contributions: I welcome suggestions for material to include in this column, including news, reviews, open problems, tutorials and surveys, either exposing the community to new and interesting topics, or providing new insight on well-studied topics by organizing them in new ways.

Recent Algorithmic Advances in Population Protocols

Dan Alistarh
Institute of Science and Technology Austria
dan.alistarh@ist.ac.at

Rati Gelashvili
University of Toronto, Canada
gelash@cs.toronto.edu

1 Background

Population protocols are a popular model of distributed computing, introduced by Angluin, Aspnes, Diamadi, Fischer, and Peralta [6] a little over a decade ago. In the meantime, the model has proved a useful abstraction for modeling various settings, from wireless sensor networks [35, 26], to gene regulatory networks [17], and chemical reaction networks [21]. In a nutshell, a population protocol consists of n agents with limited local state that interact randomly in pairs, according to an underlying communication graph, and cooperate to collectively compute global predicates. From a theoretical prospective, population protocols, with the restricted communication and computational power, are probably one of the simplest distributed model one can imagine. Perhaps surprisingly though, solutions to many classical distributed tasks are still possible. Moreover, these solutions often rely on interesting algorithmic ideas for design and interesting probabilistic techniques for analysis, while known lower bound results revolve around complex combinatorial arguments.

Part of the practical motivation behind the study of population protocols comes from an intriguing line of applied research, showing that some population protocols can be implemented at the level of synthetic DNA molecules [22], and that some natural protocols are equivalent to computational tasks solved by living cells in order to function correctly [19]. Inspired by recent developments in DNA programming, an extensive series of papers, across different communities, has examined the computability and complexity characteristics of population protocols.

Each agent in a population protocol is an identical state machine. A *configuration* captures the "global state" of the system at any given time, which can be entirely described by the number of agents in each state. The protocol starts in some initial configuration and computation happens through pairwise interactions, where both interacting agents observe each other's state and update their states according to a deterministic state transition function. This transition function is fixed and constitutes the protocol that we design. In practical applications, different pairs of agents can interact in parallel. A standard approach to facilitate analysis (reminiscent of linearizability [31]) is to consider interactions happening sequentially, one-by-one.

The first classic cost measure for population protocols is called *parallel time*: for any given sequence of interactions, it is defined as the number of interactions divided by n. This can be interpreted as the average number of interactions per agent. The other key measure is the *state complexity* of the population protocol, defined as the number of *distinct states* that each agent can internally represent. This second measure is particularly important in practice: for instance, In DNA implementations, the more states a protocol implements, the higher the likelihood of a *leak*, which is roughly an undesirable phenomenon which causes spurious creation of states following an interaction [38, 3].

These two cost measures introduce non-trivial trade-offs between the *time* in which a task can be solved, the *space* budget needed for the solution, and the *types of guarantees* that can be provided by the protocol. Providing a precise handle on these trade-offs has proven to be an exciting and technically-challenging quest, which perhaps explains the growing interest in this area.

Goals. The particular line of recent work we focus in this article is considering population protocols with *super-constant* number of states: that is, the number of states is allowed to be parametrized on the number of agents n present in the population. Adding this possibility is a departure from the original definition of the model [6], yet has proven to be fertile ground for algorithmic research, and may not be entirely impractical.

Since this area is evolving rapidly—during the writing of this article, several new papers have appeared on this topic—we do not aim to provide an authoritative survey of this area. Instead, we will focus on implementations of a limited number of fundamental constructs, and then illustrate their use as modules in more complex algorithms.

More precisely, we will begin by giving an overview of the most popular settings and tasks in population protocols, followed by a simple example super-constant state protocol for the classic leader election task. Then, we will describe three basic building blocks: *phase clocks*, *synthetic coins*, and *population splitting*. Lately, these techniques have become essential tools for designing population protocols. In particular, there has been a lot of progress by independent groups of researchers, leading to a knowledge accumulation instrumental to a number of recent results. We finish by providing examples of these techniques in the context of more complex protocols, solving leader election and exact majority.

2 Shades of Population Protocols

2.1 Communication

The sequence of interactions is determined by a *scheduler* that satisfies certain fairness requirements. A *weakly fair* scheduler must ensure that each agent interacts infinitely often, while a *globally fair* scheduler must ensure that any configuration that is reachable (via a sequence of interactions) infinitely often is indeed reached infinitely often.

Most of the time, the scheduler is assumed to be probabilistic, selecting a pair from a distribution on the edges of an underlying communication graph. Usually, the communication graph is a clique and the distribution is uniform. This setting is somewhat similar to global fairness, but enables reasoning about the time complexity of the protocol, which is a big advantage. Some previous studies, e.g. [26, 33, 37], have considered different communication graphs.

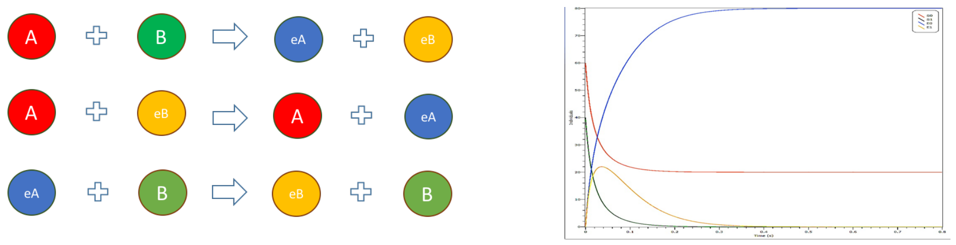

Figure 1: The three reactions defining the four-state exact majority algorithm [26] (left), and an illustration of the node state densities over time (right). Nodes start in either A (55%, red) or B state (45%, green), and proceed to interact. Notice that after a while the initial minority B state (green) is depleted, and then the remaining A nodes proceed to deplete any remaining eB nodes. The resulting population has converged to only A and eA node types, corresponding to the initial majority of A.

2.2 Tasks

In addition to the transition function, a designer of the population protocol also defines the output mapping from states to outputs. A distributed *task* in population protocols determines (a) the permissible initial configurations and (b) a logical predicate that, based on an initial configuration, describes whether a given configuration satisfies the correct output requirements.

In the *majority* task, every agents starts in one of the two input states. The output mapping associated with the protocol should map every state to a binary output representing which input state had majority in the initial configuration. The predicate then dictates that all agents be in states mapped to the correct output. Please see Figure 1 for an illustration of an algorithm solving this task, and of its execution.

In the leader election task, all agents start in the same state (usually *leader*), and every state is mapped to a binary indicator of being a leader or not. The predicate dictates that only configurations with a single leader satisfy output requirements.

Some other examples of tasks studied in population protocols that we will not consider here in detail are plurality [27], counting [12] and naming [18].

2.3 Computation

The most conservative setting requires all agents *stabilize* to a configuration, representing the output of the computation, such that all future configurations satisfy the correct output requirement. Stabilization time (defined when the scheduler is probabilistic) is the expected parallel time from the initial configuration to the stable configuration.

The stable configuration is the first moment after which every reachable configuration (regardless of subsequent interactions) is guaranteed to satisfy the output predicate. However, given any sequence of interactions leading to the stable configuration, prior configurations could also satisfy the predicate. We say that a protocol has *converged* when it reaches a configuration which satisfies the output requirements. By definition, stabilization occurs no earlier than convergence. In a configuration where the protocol has converged but not yet stabilized, there exists a non-zero probability of divergence, which does not get realized in the given sequence of interactions to the stable configuration.

Stabilization is a stronger requirement, but fast convergence time can sometimes be sufficiently good for practical purposes. More importantly, it can allow protocols to have smaller state complexity, bypassing known lower bounds for stabilization [25, 1, 13, 2]. Similarly, it is reasonable to relax the exact computation requirement by allowing a low probability that the protocol never converges to the correct output. A well-known protocol for approximate majority uses just three states and stabilizes in a poly-logarithmic time [8], but the same is impossible for exact computation with less than $\log \log n/2$ states [1].

In certain cases it is possible to combine different types of protocols. Suppose P_1 is a protocol that with high probability converges fast to the correct output, or some agent can detect the failure. Then, we can combine P_1 with an always correct, slow to stabilize protocol P_2 with a few states. The state complexity of the resulting protocol is the product of state complexities of P_1 and P_2.

2.4 Other settings

In *self-stabilizing* population protocols, the requirement is to stabilize from an arbitrary initial configuration. Research on self-stabilizing population protocols [10] has shown that the requirement might be too strict. Intuitively, this is because individual agents are not aware of the total number of agents in the system, and thus, it is possible to combine multiple valid output configurations for a smaller subset of agents as an input configuration. In fact, [10] shows the impossibility of solving leader election using such a partitioning argument.

In [36], Sudo et al. found a really nice way around this problem by definition a concept of *loosely-stabilizing* population protocols, where the requirement is that from any configuration, the protocol must reach a correct output configuration as fast as possible (which is the usual notion of stabilization time), and then remain correct for as long as possible (which is called the *holding time*). Typical goal is polylogarithmic stabilization time and exponential holding time.

Another direction is to investigate the robustness of population protocols to faulty interactions. This includes the study of *leaderless population protocols* [13], as well as computing correctly in the presence of leaks [4], that is, roughly, spurious reactions.

3 A Simple Leader Election Protocol

A progression of deep technical results [24, 21] culminated in Doty and Soloveichik showing that a stable solution to leader election is impossible in sublinear expected time for protocols which are restricted to a *constant* number of states per agent [25]. Concurrently, we showed that this lower bound can be circumvented, by designing a simple algorithm called "Leader-Minion", which solves leader election in $O(\log^3 n)$ parallel time and requires $O(\log^3 n)$ states per agent [5].

Algorithm Description. To describe this algorithm, we start from the classic "tournament" protocol: every agent starts as a potential leader, and whenever two leaders interact, one drops out of contention. The main drawback of this protocol is that, once only a constant number of potential leaders remain, they take a long time to interact, leading to $\Omega(n)$ stabilization time. To overcome this problem, we introduced a propagation mechanism, which adds a "seeding" mechanism, based on the number of interactions a leader has performed so far. Contenders compete by comparing their seeding, and the agents who drop out of contention become "minions" for the corresponding leader, assuming the identity of their victor. Thus, a minion representing an agent of higher seeding can cause agents still in contention, but with lower seeding, to drop out.

More precisely, we let the states be integers, where all agents start in state 0. A non-negative integer $s \geq 0$ represents the state of a contender with seeding s, and a negative integer $s < 0$ represents the state of a minion that knows about the existence of a contender with seeding at least $-s$. In an interaction between two contenders with seedings s_1 and s_2, if $0 \leq s_1 < s_2$, the new states will be $-s_2$ and $s_2 + 1$, i.e. the contender with lower seeding will become a minion.

The key ideas are that 1) a contender that does not become a minion during an interaction always increases its seeding, and that 2) each minion keeps track of the maximal seeding of a contender in the system that it has been aware of. Hence, if a minion in a state $s_1 < 0$ meets another agent in a state s_2 with $|s_2| > |s_1|$, then the minion will change its state to $-|s_2|$.

Finally, we combine a capped version of the above protocol with states in $[-m, m]$ for some $m \in \Theta(\log^3 n)$ with a slow, always correct protocol executed among contenders with seeding m. This is similar to the method described in Section 2.3 and results in an always correct protocol with $O(\log^3 n)$ state complexity and stabilization time.

4 Population Protocol Design Toolkit

4.1 Phase Clocks

A *phase clock* is a shared object which allows the agents to have an (approximate) common notion of time; in the context of population protocols, nodes often use phase clocks to collectively count time in phases of $\Theta(n \log n)$ interactions, with bounded *skew*, by which we mean the maximum clock value difference between two nodes. More precisely, the phase clock ensures that all agents will be in the same phase during at least $\Theta(\log n)$ interactions of each agent. Angluin, Aspnes, Eisenstat and Ruppert [9] introduced phase clocks as a critical component to perform generic register simulations using population protocols. Their construction used constant states per node, but required the existence of an initial *unique leader* to coordinate the exchange of information among agents. In addition to practical concerns, this is problematic as leader election is itself a non-trivial task to solve. Partly for this reason, in the following decade, phase clocks had rarely been used in the design of population protocols which optimize for speed of convergence.

Two recent papers [2, 28] approached the issue of efficiently implementing phase clocks in different ways. In joint work with James Aspnes [2], we designed a *leaderless phase clock* with $O(\log n)$ states, where the synchronization between nodes is achieved via on a connection to a power of two choices load balancing process [11].[1]

Gąsieniec and Stachowiak [28] made a breakthrough on state-efficient phase clocks, by showing that the underlying mechanism behind the construction in [9] could also work with a junta of $n^{1-\epsilon}$ leaders, for $\epsilon > 0$. This was suggested as a possibility in [9], but its technical realization had remained a long-standing open problem. The phase clock construction of [28] uses $O(\log \log n)$ states. Interestingly, only *constantly many states* are used for the clock, while the $O(\log \log n)$ states are used for obtaining the desired junta of leaders. In a testament to the rapid developments in this field, in [14], Berenbrink et al. have already designed a scheme which enables recycling these states after the junta is elected. [14] also provides an elegant, simplified exposition of Gąsieniec and Stachowiak's phase clock.

[1] Roughly, each node maintains a local counter representing its current clock value; whenever two nodes interact, the one with a lower counter value increments its counter by one, while the other keeps its counter unchanged. Additional standard tricks can be applied to maintain a bounded state space size.

Recently, in [32], Kosowski and Uznanski built upon this idea by constructing a hierarchy of phase clocks, allowing agents to synchronize in phases of $\Theta(n \log^k n)$ interactions, where k is a parameter.

4.2 Synthetic Coins

The state transitions in population protocols are usually deterministic. While this is certainly beneficial for protocol simplicity, it can be a restricting factor for algorithm design. For instance, [37] contains a protocol for a loosely-stabilizing leader election that would work if agents had access to a source of random bits. In general, it is well-understood that randomization often can unlock considerable power in designing better and simpler algorithms, both in standard sequential and distributed models [34].

In a joint work with Aspnes, Eisenstat and Rivest [1], we noticed that even though state transitions are deterministic, there is still a source of randomness in population protocols: the *random scheduler*. We introduced a new *synthetic coin* technique, that allows agents to extract the randomness from the scheduler and simulate almost fair coin flips at the cost of modest increase in the state complexity.[2]

Our construction works as follows. Each state has a single additional *coin* bit (doubling the state complexity), which is flipped at every interaction. The bit-string of all coin bits (considered in some fixed order of agents) behaves as a random walk on a hypercube, and can be analyzed using probabilistic tools. We used McDiarmid's inequality to show that the number of agents with coin bits equal to 0 and 1 will be roughly equal after constant parallel time with high probability. Hence, after a short mixing stage which is easy to administer (e.g. by disregarding the first 4 interactions as in [1]), each agent can use the coin bit of its interaction partner as an almost-fair coin.

In [16], Berenbrink, Kaaser, Kling and Otterbach extended the above mechanism in two important ways. First, they proved that by letting the mixing stage last longer, it is possible to obtain a tighter bound on the number of 0 and 1 coin bits in the system, improving the guarantees of the almost-fair synthetic coin. Second, they introduced a technique that uses almost-fair synthetic coin flips to mark a given fraction of the agents. Then, whether or not an agent meets a marked agent can be used as a synthetic coin which has a specific desired bias (based on the fraction of the marked agents). Since marking is done once, this improves over a naive approach of simulating each coin flip with a specific desired bias by flipping multiple almost-fair coins.

Another development regarding biased synthetic coin flips was by Gąsieniec, Stachowiak and Uznański [29]. Building upon junta-based leader election construction [28], they show how to generate a spectrum of $O(\log \log n)$ coins with different biases. Junta members are spread across multiple levels, and flipping ℓ-th is performed by checking if the interaction partner is at ℓ-th or higher level.

4.3 Splitting the Population

Recall that in the Leader-Minion protocol [5], m states (0 through $m-1$) are used by contenders and $m-1$ states are used by minions. Suppose we split the 0 state into $+0$ (contender) and -0 (minion) states. Then the protocol can be viewed as having a contender/minion role indicator (i.e. plus/minus sign of the state value) and otherwise using the same $\log m$ states for agents in both

[2]Cardelli, Kwiatkowska, and Laurenti [20] concurrently introduced a similar construction, although their focus is on computability, rather than complexity and convergence speed.

roles. Similarly, in the leader election protocol in [1] the state of each agent has a more refined indicator for the stage of the protocol and the role of the agent, but the rest of the state space is shared among agents with different indicators. This is only possible when the roles are mutually exclusive: e.g. in Leader-Minion protocol, every agent is always either a contender or a minion, but not both.

The agents in different stages of the algorithm rarely ever interact with each other, and the leader-minion mechanism was rather ad-hoc. In [30], Ghaffari and Parter explicitly split the population into distinct roles, with their results critically depending on the fact that agents in different roles co-exist, but execute different algorithms in terms of state-transitions. In [2], we took this idea even further, splitting the population into worker and clock agents, where workers executed majority protocol among themselves, while clocks executed the leaderless phase clock. The notification about a phase ending or starting was spread through the whole population via rumor spreading. Our majority protocol in [2] has state complexity $O(\log n)$, while relying on phase clock and majority gadgets that each require $O(\log n)$ states. Without population splitting, the resulting protocol would have had state complexity $\Theta(\log^2 n)$. In is worth noting in a different protocol in [2], we used synthetic coins to break ties and split the agents into different roles.

Other papers have since demonstrated the power of this technique. In [29], the population is split in three different roles that interact in a relatively complex manner, and [15] provides an example of a rather delicate splitting of state space into two roles that resemble clocks and workers from [2].

We note that population splitting is somewhat similar to the problem of *task allocation* in insect populations, e.g. [23], although the setting and techniques used are rather different.

5 Applications

We now illustrate how the building blocks we described above have been combined in non-trivial ways to build more complex population protocols.

5.1 Protocols for Exact Majority

As discussed in the previous section, population splitting is key in both [2] and [15] for obtaining population protocols for stable majority computation with $O(\log n)$ states, which is optimal by [2], under combinatorial assumptions on the protocol structure. In both cases, the population is split in worker and clock agents, but these roles have different meanings.

In [2], clock agents directly implement a phase clock construction between themselves. Worker agents alternate between $\log n$ cancellation phases (where agents from the minority and the majority interact and cancel each other out) and doubling phases (where agents try to spread their opinion). Critically, the exact alternation of these phases is facilitated by the phase clocks. The basic idea behind this protocol was introduced in [7], for a setting with an initial unique leader; the underlying dynamics is that each iteration of cancellation + doubling increases the advantage of the majority opinion by a factor of two. The protocol as constructed stabilizes with high probability in $O(\log^2 n)$ parallel time, but admits a low probability of error, e.g. due to a phase clock or workers getting out of sync. Because these errors can always be detected by some agent, using another common trick described earlier, the protocol is then combined with a slow, always correct 4-state backup protocol [26, 33]. This yields an always correct protocol with $O(\log n)$ states and $O(\log^2 n)$ stabilization time.

The protocol presented in [15] achieves $O(\log^{5/3} n)$ stabilization time by cleverly fusing together multiple consecutive phases of [2] into an *epoch*, and building a mechanism that can tolerate agents being in phases that are close to each other as opposed to exactly the same. This necessitates keeping track of both epoch and phase counters, and the authors delegate keeping track of some of these counters to the clock agents. This is the reason why in this protocol, "clock" agents are not implementing an actual independent phase clock.

5.2 Protocols for Leader Election

Since leader election is the task which triggered the development of the synthetic coin technique in [1], it may come as no surprise that the best known protocols [29, 14] all rely on this mechanism. Both of these protocols (as well as [28]) use $O(\log \log n)$ states, the protocol of [29] has (expected) stabilization time $O(\log n \log \log n)$ and the protocol of [14] has (with high probability) stabilization time $O(\log^2 n / \log \log \log n)$. Even protocols with $O(\log n)$ state complexity [2, 16] utilize synthetic coins. Generally, the better the protocol, the more sophisticated the details, so we show example uses of synthetic coins in earlier protocols, which provide a more modular illustration of the underlying ideas.

The way in which synthetic coins were used originally in [1] was motivated by the fact that in the leader-minion algorithm, a contender required $\Theta(\log n)$ larger seeding to reliably eliminate another contender. Basically, this is required for its minions to have a good likelihood of reaching the lower-seeded contender before its seeding increases. If we could slow down the process of increasing seeding, then there would be enough time for the rumor to spread through, and eliminate more contenders. This would make the seeding of the last standing contender lower, decreasing the state complexity. The protocol in [1] achieved this by allowing the contenders to increase seeding only if they have flipped a certain number of consecutive almost-fair coins as heads. As explained earlier, the construction of a synthetic coin with bias of [16] improves this by replacing multiple almost-fair coin flips (and the state overhead for tracking them) with a single biased coin flip.

Another way to utilize synthetic coins, which has become prevalent in recent protocols, is to use the outcome of the coin to decide whether a contender will attempt to drop out of contention. The key here is to ensure that not all contenders drop out. Hence, before proceeding, a contender that is trying to drop out of contention must observe a contender that is not trying to drop out. A simple example of this can also be found in the leader election protocol in [1], which has the following lottery stage: each contender uses synthetic coin flips to get a random seeding, information about the maximum seeding in the system is then propagated using rumor spreading, and all contenders with lower seeding drop out.

Finally, phase clocks have been crucial in achieving the optimal state complexity of $O(\log \log n)$ for stable leader election in [28], the construction that both [29] and [14] build upon. Once the phase clock is set up in [28], it becomes easy to eliminate roughly half of the contenders using synthetic coins as described in the previous paragraph. When a new phase begins, a contender flips an almost-fair coin to decide whether it will try to drop out in this phase, and a phase consists of enough interactions for all agents to learn if any contender is not trying to drop out (eliminating all contenders that are trying).

6 Conclusion

Over the past decade, population protocols have become one of the rapidly-developing, exciting areas in distributed computing. As more research attention is applied to the fundamental algorithmic problems in this space, we discover new tools and techniques for examining the fundamental trade-offs between speed, space, and convergence guarantees in population protocols. Although these tools tend to become more complex, they can often be reduced to elegant, non-trivial combinations of basic building blocks.

The goal of this article has been to perform a non-exhaustive survey of recent developments in super-constant state space population protocols, and highlight some interesting uses of basic building blocks, notably phase clocks, synthetic coins, and population splitting. While the pace of progress in this area is brisk, we wanted to convey the fact that even relatively complex protocols can be easily deconstructed at an intuitive level, and that the "entry price" into the area is relatively low.

References

[1] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. Time-space trade-offs in population protocols. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms*, (SODA), pages 2560–2579, 2017.

[2] Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms*, (SODA), pages 2221–2239, 2018.

[3] Dan Alistarh, Bartłomiej Dudek, Adrian Kosowski, David Soloveichik, and Przemysław Uznański. Robust detection in leak-prone population protocols. In *International Conference on DNA-Based Computers*, pages 155–171. Springer, 2017.

[4] Dan Alistarh, Bartłomiej Dudek, Adrian Kosowski, David Soloveichik, and Przemysław Uznański. Robust detection in leak-prone population protocols. In *Proceedings of the 23rd International Conference on DNA Computing and Molecular Programming*, pages 155–171, 2017.

[5] Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming*, (ICALP), pages 479–491, 2015.

[6] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed computing*, 18(4):235–253, 2006.

[7] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, 2008.

[8] Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.

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

[10] Dana Angluin, James Aspnes, Michael J Fischer, and Hong Jiang. Self-stabilizing population protocols. *ACM Transactions on Autonomous and Adaptive Systems*, 3(4):13:1–13:28, 2008.

[11] Yossi Azar, Andrei Z Broder, Anna R Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.

[12] Joffroy Beauquier, Janna Burman, Simon Clavière, and Devan Sohier. Space-optimal counting in population protocols. In *Proceedings of the 29th International Symposium on Distributed Computing*, (DISC), pages 631–646, 2015.

[13] Amanda Belleville, David Doty, and David Soloveichik. Hardness of computing and approximating predicates and functions with leaderless population protocols. In *Proceedings of the 44nd International Colloquium on Automata, Languages, and Programming*, (ICALP), pages 141:1–141:14, 2017.

[14] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. Majority & stabilization in population protocols. *arXiv preprint arXiv:1805.04586*, 2018.

[15] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A population protocol for exact majority with $o(\log^{5/3} n)$ stabilization time and asymptotically optimal number of states. *arXiv preprint arXiv:1805.05157*, 2018.

[16] Petra Berenbrink, Dominik Kaaser, Peter Kling, and Lena Otterbach. Simple and efficient leader election. In *1st Symposium on Simplicity in Algorithms*, (SOSA), 2018.

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

[18] Janna Burman, Joffroy Beauquier, and Devan Sohier. Space-optimal naming in population protocols. *hal preprint hal:01790517*, 2018.

[19] Luca Cardelli and Attila Csikász-Nagy. The cell cycle switch computes approximate majority. *Nature Scientific Reports*, 2:656:1–656:9, 2012.

[20] Luca Cardelli, Marta Kwiatkowska, and Luca Laurenti. Programming discrete distributions with chemical reaction networks. In *International Conference on DNA-Based Computers*, pages 35–51. Springer, 2016.

[21] Ho-Lin Chen, Rachel Cummings, David Doty, and David Soloveichik. Speed faults in computation by chemical reaction networks. *Distributed Computing*, 30(5):373âĂŞ390, 2017.

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

[23] Alejandro Cornejo, Anna Dornhaus, Nancy Lynch, and Radhika Nagpal. Task allocation in ant colonies. In *International Symposium on Distributed Computing*, pages 46–60. Springer, 2014.

[24] David Doty. Timing in chemical reaction networks. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms*, (SODA), pages 772–784, 2014.

[25] David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018.

[26] Moez Draief and Milan Vojnovic. Convergence speed of binary interval consensus. *SIAM Journal on Control and Optimization*, 50(3):1087–1109, 2012.

[27] Leszek Gąsieniec, David Hamilton, Russell Martin, Paul G Spirakis, and Grzegorz Stachowiak. Deterministic population protocols for exact majority and plurality. In *Proceedings of the 20th Conference on Principles of Distributed Systems*, (OPODIS), pages 14:1–14:14, 2016.

[28] Leszek Gąsieniec and Grzegorz Stachowiak. Fast space optimal leader election in population protocols. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms*, (SODA), pages 2653–2667, 2018.

[29] Leszek Gąsieniec, Grzegorz Stachowiak, and Przemysław Uznański. Almost logarithmic-time space optimal leader election in population protocols. *arXiv preprint arXiv:1802.06867*, 2018.

[30] Mohsen Ghaffari and Merav Parter. A polylogarithmic gossip algorithm for plurality consensus. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing*, (PODC), pages 117–126, 2016.

[31] Maurice P Herlihy and Jeannette M Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492, 1990.

[32] Adrian Kosowski and Przemyslaw Uznanski. Population protocols made easy. *arXiv preprint arxiv:1802.06872*, 2018.

[33] George B Mertzios, Sotiris E Nikoletseas, Christoforos L Raptopoulos, and Paul G Spirakis. Determining majority in networks with local interactions and very small local memory. *Distributed Computing*, 30(1):1–16, 2017.

[34] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[35] Etienne Perron, Dinkar Vasudevan, and Milan Vojnovic. Using three states for binary consensus on complete graphs. In *Proceedings of the 28th IEEE Conference on Computer Communications*, INFOCOM '09, pages 2527–2535, 2009.

[36] Yuichi Sudo, Junya Nakamura, Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Loosely-stabilizing leader election in a population protocol model. *Theoretical Computer Science*, 444:100–112, 2012.

[37] Yuichi Sudo, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Loosely-stabilizing leader election on arbitrary graphs in population protocols. In *Proceedings of the 18th International Conference on Principles of Distributed Systems*, (OPODIS), pages 339–354, 2014.

[38] Chris Thachuk, Erik Winfree, and David Soloveichik. Leakless dna strand displacement systems. In *Proceedings of the 21st International Conference on DNA Computing and Molecular Programming*, DNA21, pages 133–153, 2015.