Distributed computing inspired by biology

Matthias Függer, Thomas Nowak, Kerian Thuillier

Distributed Computing Inspired by Biology

Matthias Függer*[1], Thomas Nowak*[1,2], and Kerian Thuillier[1]

[1]Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, Gif-sur-Yvette, France
[2]Institut Universitaire de France, Paris, France

Abstract

Biological systems are mastering the art of composing cells into colonies, tissues, and organisms. This article reviews striking similarities and differences between such biological systems and distributed computing systems, where computational units are composed to form larger systems with the goal of increasing computational power, enhancing system robustness, or overcoming spatial distances.

A problem that recurs in many contexts in distributed systems is obtaining a consistent view of part of the system by its agents. Such problems, known as agreement problems in distributed computing, have been extensively studied across different computational models, varying, for example, in the extent to which the network is stable or dynamic.

Motivated by the importance of agreement problems, we discuss examples ranging from simple to more complex cases, the latter in the context of optimization: agents solving graph optimization problems, searching for optima in arbitrary loss landscapes, and applying gradient-based techniques closely related to widely adopted artificial neural networks.

We then discuss the reverse direction: distributed systems implemented with biological material. In particular, we detail a theoretical distributed computing model and algorithm targeted toward implementation in bacterial populations.

We conclude with an outlook on what we consider the beginning of a promising intersection between distributed computing and biology, highlighting opportunities for both understanding natural systems and engineering novel distributed systems, both biological and in silico.

Keywords: distributed computing; biological analogies; agreement; optimization; graph algorithms; biological distributed systems

*Corresponding authors: mfuegger@lmf.cnrs.fr, thomas@thomasnowak.net

1 Introduction

Developing technical solutions by seeking inspiration from existing biological solutions has a long history with a prominent example being the history of aviation inspired by the flight of birds. Like mechanical engineering, computer science has drawn inspiration from biology, and conversely, computer science has developed computational methods that are now state-of-the-art in biological research.

Previous reviews have examined the relation between computer science and biology, highlighting the inherent distributed nature of biological entities and systems. Navlakha and Bar-Joseph [1] identify a convergence between the disciplines of computer science and biology: while early interdisciplinary work was primarily in processing of biological data, they see a potential for biological models obtained with a computer science perspective and algorithms in computer science that stem from a more detailed view on biological systems. They discuss examples from the domains of coordination, computation on networks, and vision. In a later review [2], Navlakha and Bar-Joseph further detail the distributed nature of biological systems from a communication and network perspective: they mention that, presumably due to the historical motivation of the theory of distributed computing by computer systems, system models with relatively powerful communication means have been studied. For example, it is often assumed that a computing node can send arbitrarily long binary messages addressed to specific nodes. Theoretical work on restricted forms of communication [3, 4, 5, 6] identified collision problems in wireless networks and was later extended to bio-inspired contexts. Navlakha and Bar-Joseph further observe that networks considered in theoretical settings are often fully centralized or distributed, while biological systems often combine both features.

In their review [7], Feinerman and Korman emphasize distributed examples and provide a detailed discussion of system models and algorithms. In distributed computing, a model is understood as the laws[1] the computational device running the algorithm and the surrounding environment follow. An example of such a law is the assumption that every message sent is also received within a fixed amount of time. This law can either be present or absent in a model. They propose classifying research into known and unknown models and algorithms, thereby making the underlying assumptions in this interdisciplinary domain explicit.

The aims of this review are fourfold:

(i) Through several examples, we show how distributed computing and optimization problems can be seen as closely related problems of increasing generality, with solutions inspired by biology. By the sheer amount of work on biophysical models and distributed computing, these examples are neither com-

[1] While in distributed computing laws are referred to as system assumptions, we use the term law to emphasize the close relation to physical laws in natural sciences. However, unlike typical physical laws, the assumptions in distributed computing typically do not allow one to predict unique future states from the current system state, but rather only constrain the space of future states.

plete nor do they cover all aspects of the two topics. We chose to focus on examples that consider convergence towards agreement within a population, as this problem is often found as a sub-problem of more complex problems.

(ii) While most examples illustrate the transfer of ideas from biology to computer science, we also discuss how these cases can pave the way for new insights in biology through subsequent transfer back to the biological domain. In Section 5, we further present an example for the transfer in the opposite direction, from distributed computing to synthetic biology, a field that is becoming increasingly intertwined with distributed computing.

(iii) We would like to emphasize, through the discussed examples, the importance of both similarities and differences of models in distributed computing and biology. While one might intuitively wish for close models in both domains, this may be counterproductive in several of the discussed examples, while important for others. For example we will stress that proximity of computational models to mechanistic biological models is not necessary and potentially limiting in optimization techniques like particle swarm optimization and genetic algorithms as well as in the design of artificial neural networks: particle swarm optimization clearly does not capture the behavior of bee swarms, genetic optimization falls short in modeling correctly homologous recombination between DNA strands, and artificial neurons show threshold behavior that we demonstrate in Section 4 to be not the case in mechanistic neural models. Indeed, the risk of directly copying models across disciplines in interdisciplinary research is not specific to biology and computer science. Success stories such as aviation demonstrate that copying, or remaining too close to the initial inspiration, may not provide feasible solutions; see, e.g., the wing-flapping solutions by Leonardo da Vinci, which did not prevail. While one may think of future technologies in synthetic biology that may render da Vinci's approach feasible, currently available technologies clearly favor less flexible constructions and it would be counterproductive to dismiss them due to their deviation from the original biological model of flight.

Conversely, computational models designed for application in biology must necessarily remain close abstractions of the underlying mechanistic biological models. We illustrate this in Section 5, where a distributed algorithm has been designed for the implementation in engineered bacteria. In this context, a model that neglects key aspects such as non-constant populations, may yield results that are not transferable to real-world implementations.

(iv) A final intention of this review is to add a hands-on, coding component to the exciting world of distributed computing and biology: We aim to share our enthusiasm for the problems and algorithms in this field through concise, easily adaptable Python code that readers can experiment with. The code is available at `https://github.com/BioDisCo/dc_bio`.

Organization of the review. We begin in Section 2 with basic and conceptually simple forms of coordination among agents of a biological or computational distributed system—specifically, reaching agreement on a common value. The discussed algorithms are inspired by simple physical and biologi-

cal entities that update their states to lie within the range of states that they received from other agents. We emphasize two variants of agreement that we deem particularly closely related to biological systems: asymptotic agreement and synchronization in time.

In Section 3 we continue with problems of increased complexity: while the goal in the former section was to agree on a value, the problem discussed here is to agree on an optimal value. We begin with specific biology-related optimization problems on graphs in Section 3.1, followed by optimization problems for general loss functions (Section 3.2) and for differentiable loss functions (Section 3.3). For the latter, artificial neuron models play a central role.

In Section 4, we compare a mechanistic biological model of a neuron with the abstract model typically used in artificial neural networks We show that the latter falls short to explain properties of the biological one, including the absence of a fixed threshold. Nonetheless—or perhaps precisely because of this simplification—the artificial model is widely and effectively used in computational contexts.

Finally, while previous sections have addressed the transfer from biology to distributed computing, Section 5 presents transfer in the opposite direction.

We conclude in Section 6.

2 Agreement Among Agents

The moment a computational or biological system is distributed among multiple computational or biological units, the problem of coordination arises. Entities communicate to coordinate their computation and actions (i.e., their behavior). Coordination becomes particularly difficult if the computational infrastructure is not static. Changes can be due to faults of entities, which include silent failing and divergence from the algorithm of single nodes and communication links, as well as changes in the communication infrastructure. Examples in the computational world for the former are hardware and software faults, as well as adversarial intrusion, while examples for the latter are peer-to-peer networks, wireless networks, and other systems with unstable communication. Biological analogs include malfunctioning cells in a multicellular organism, deviant behavior of bacteria within a population (which may threaten the population's survival), or external intrusions such as infection of a multicellular organism by pathogenic bacteria.

One of the most basic coordination problems is agreeing on a common value. Such agreement can take the form of explicitly agreeing on the value of a local variable [8], synchronizing clocks across a network [9, 10, 11], or coordinating the generation of a digital clock signal by different units on a chip [12, 13].

While exact agreement is crucial in computational distributed systems—for example, when keeping a bit-wise identical consistent bank account state or shopping cart state among multiple distributed replicas—more relaxed forms of agreement are observed in biological systems. Examples include collective directional decisions in schools of fish [14] or flocks of birds [15], where individuals

coordinate to prevent collisions and move in a common direction.

Similarly, relaxed versions of agreement have been studied in computer science—for example, motivated by distributed control, where the controller's output is not required to be identical among replicas, and clock synchronization, where agents repeatedly agree on an approximately similar time.

Asymptotic agreement. Closest to biological systems might be the problem of reaching asymptotic agreement. Here, a system of agents communicating in discrete rounds is considered. Each agent starts with an initial local value in $\mathbb{R}^n$, known only to itself. During each round, agents broadcast messages, and the environment determines which agents receive which messages. Agents then update their states and values based on their current local information and the messages received. An algorithm—specifying how messages are broadcast and how states and outputs are updated—is said to solve asymptotic agreement if all agents' output values converge to a common value over time. Desirable properties of such algorithms are robustness across a wide range of environments, low computational and memory costs, and fast convergence.

An environment can be described by the set of possible communication graphs that may occur in a round of communication. In such a communication graph, an edge from agent a to agent b exists if the message broadcast by a is received by b during that round. The structures of the graphs are naturally application-dependent: an abstract model for a bird flock—where messages correspond to one bird observing another—will presumably differ from a model of cars communicating via peer-to-peer networks.

An approach often taken in distributed computing is to over-approximate the application environment, by a more adversarial one that is simpler to specify and analyze. For example, environmental instability can be captured by allowing the environment to change the communication graph in each round. In such highly dynamic networks it has been shown that for the problem to be solvable, each graph must contain a spanning tree [16], i.e., a tree within which all the graph's nodes are reachable from a single root, although the tree can change from round to round. Figure 1a illustrates an example sequence of communication graphs that switch every round in a system of 10 agents.

A simple yet widely adopted update scheme sets each agent's value to the average of the values it receives. Despite its simplicity, repeatedly averaging and distributing values can resemble biological phenomena, ranging from social influence models [17] to bird flocks [18]. However, such models have important limitations in their applicability. For instance, while bird flocks exhibit convergence in position and direction among distant individuals, nearby birds do not converge to a single spatial point. Similarly, formation control algorithms for drones use attractive behavior at long range but apply virtual repulsive forces at short range to prevent collisions [19, 20]. Despite these limitations, such simple averaging algorithms can be shown to solve the asymptotic agreement problem [21, 16] even in highly dynamic environments. In fact, such algorithms solve the problem under minimal assumptions—namely, for the largest class of

graphs for which a solution can provably exist.

Figure 1b illustrates the equal-weight averaging algorithm in action. In a system of 10 agents with random initial values, and under a sequence of communication graphs that change every round (Figure 1a), the agents repeatedly update their values by averaging their own value and all received values with equal weights. The agents' outputs are seen to be very close after 6 rounds.

Interestingly—and at first perhaps counterintuitively—an even simpler algorithm can be shown to converge faster [22, 23] by interleaving averaging updates with flooding rounds, during which agents simply forward the messages they have received. Figure 1c shows the algorithm in action with one flooding round interleaved between each averaging round. All agents have reached the same output value by round 5.

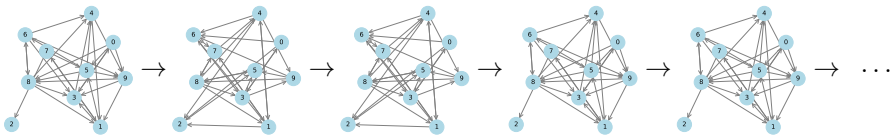
Of particular mention is an algorithm that can be shown to be optimal in convergence speed under certain conditions for one-dimensional values in $\mathbb{R}$ [23]. Instead of an unweighted average, agents update their value to the midpoint of the set of received values. The midpoint of a set S is defined as $(\max(S) + \min(S))/2$. Example runs of the midpoint algorithm, and of the midpoint algorithm with interleaved flooding rounds in random graph sequences, are shown in Figures 1d and 1e, respectively. The algorithm with interleaved flooding reaches the same output value for all agents by round 5.

Randomly generated graphs do not always hit the worst-case of convergence times for the equal-weight and midpoint algorithm. Figure 1f shows the performance of the equal-weight algorithm in a specific deterministic graph (the butterfly graph, [24, Figure 1]) in which it was proven that the algorithm exhibits a convergence time that is exponentially large in the number n of nodes. On the other hand, the midpoint algorithm interleaved with $n - 1$ flooding rounds converges in finite time, specifically after n rounds (Figure 1g).

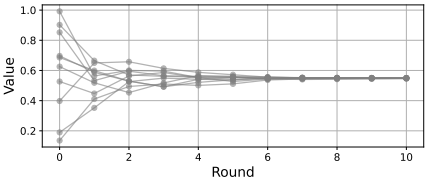
The discovery of the great speedup through interleaving flooding rounds in computational systems, opens new questions when transferring these findings back to biological systems: We are not aware of correspondences of alternating averaging and flooding in biological systems, but conjecture that due to their optimality and simplicity, such schemes may indeed be found.

While averaging algorithms like the equal-weight algorithm are well-defined for higher-dimensional values in $\mathbb{R}^d$ with $d \geq 2$, the midpoint algorithm in terms of max and min is not defined beyond scalar values. An alternative formulation of the midpoint of points in S is the midpoint on the line between the two most distant points in S. This algorithm, called *MidExtremes*, has been proven to converge fast for higher-dimensional values [25], with a convergence rate that is independent of the dimension d.

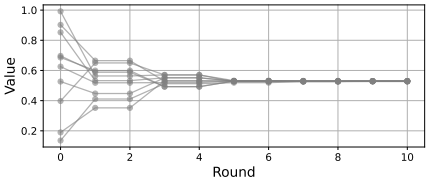
From the perspective of a biological agent, this would require determining distances between all of the agents' values it observes (e.g., positions in $\mathbb{R}^3$). However, such an approach presents challenges in both biological and artificial systems, such as drone formations. To address this, the authors studied the *ApproachExtreme* algorithm, where agents update their value to the midpoint of the line with the longest distance between themselves and a received value. Interestingly, this algorithm can be shown to converge with a speed independent



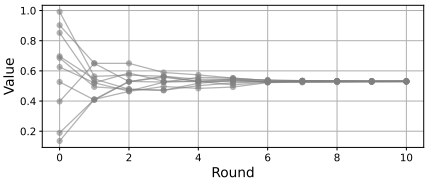
(a) Agents' output values over rounds. Initial values are randomly chosen within $[0, 1]$.



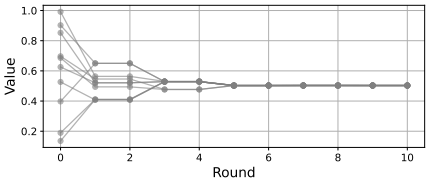
(b) Equal-weight algorithm.



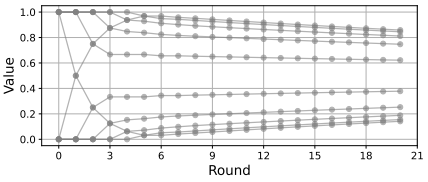
(c) Equal-weight algorithm alternating with a flooding round.



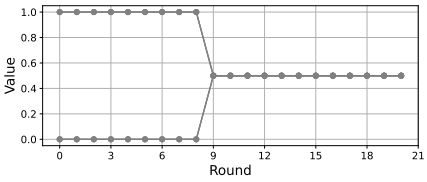
(d) Midpoint algorithm.



(e) Midpoint algorithm alternating with a flooding round.

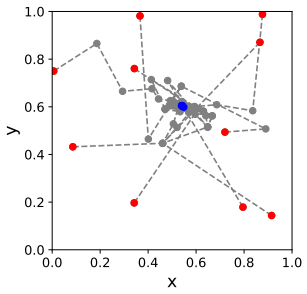


(f) Equal-weight algorithm in the butterfly graph.

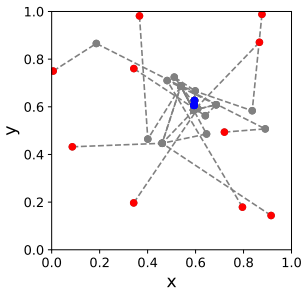


(g) Midpoint algorithm alternating with 9 flooding rounds in the butterfly graph.

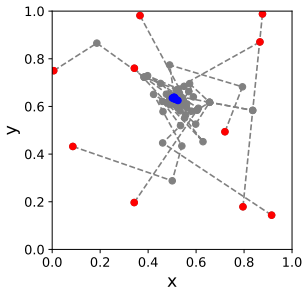
Figure 1: Asymptotic agreement among 10 agents. The communication graph changes at each round. The first 5 rooted graphs of the sequence of communication graphs are shown in **(a)**. Figures **b-e** show the agents' output values over rounds. Each agent's initial value is drawn independently from a uniform distribution on $[0, 1]$. Figures **b** and **c** show the values output by agents with the equal-weight algorithm, with and without flooding round, respectively. Figures **d** and **e** show the values output by agents with the midpoint algorithm, with and without flooding round, respectively. Figures **f** and **g** show the values output by agents with a worst-case initial-value assignment in the butterfly graph, with the equal-weight algorithm **(f)** and the midpoint algorithm with 9 flooding rounds **(g)**.



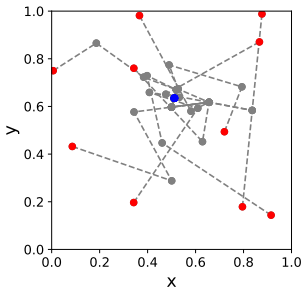
(a) *MidExtremes* algorithm.



(b) *MidExtremes* algorithm alternating with a flooding round.



(c) *ApproachExtreme* algorithm.



(d) *ApproachExtreme* algorithm alternating with a flooding round.

Figure 2: Asymptotic agreement among 10 agents in $\mathbb{R}^2$. Agents' output values over 6 rounds. Initial values are randomly chosen within $[0, 1]^2$ (in red), trajectories are shown in gray, and final values are shown in blue. All algorithms converge to close values within the 6 rounds.

of the values' dimension. The latter algorithm may very well have correspondences in biological settings: it seems plausible that an agent attempting to stay close to others tries to approach the agent furthest away from itself, adapting its speed to meet it halfway within a given time frame.

Figure 2, and Supplementary Videos S1–S4 corresponding to Figures 2a–d, show a comparison of the two algorithms and their variants with alternating flooding rounds for 6 rounds. All algorithms are seen to converge to close values within the 6 rounds.

Synchronization in time. As previously noted, averaging algorithms can also be used to synchronize agents' actions in time. For this purpose, each agent locally tracks time with its own clock, which may drift from others due to slight frequency mismatches. To counteract this drift, agents repeatedly observe other agents' clocks and adjust their own time to the average of the observed times. The details of how an agent measures another agent's time and determines the averaging weights depend on the specific model, application, and algorithm. For

example, in powerful computer networks, local clock readings may be exchanged as timestamp messages over IP networks, whereas simple agents instead send periodic pulses and measure the interval between receiving another agent's pulse and emitting their own. Care must be taken because algorithms that use only pulses have no access to the origin of the sender and pulses too close in time will be detected as a single event. Furthermore, some simple networked chips have only a single antenna and can therefore either listen or emit a pulse, but not both simultaneously.

Indeed, clock synchronization in distributed systems based on averaging algorithms has been proposed and analyzed for static [26], fault-tolerant static [9], and dynamic [27] communication settings. On the biological side, pulse-based synchronization has been observed in several systems [28], including fireflies synchronizing via light pulses [29, 30], pacemaker cells in the rabbit heart synchronizing via electrical signals [31], and the lobster cardiac ganglion [32, 33]. These systems have been modeled as abstract, algorithmic, pulse-coupled local clocks [28, 34]. Studies of their fault tolerance [33] have inspired new algorithms for simple computing devices [35].

For example, the lobster cardiac ganglion system was found to exhibit remarkable fault tolerance [33], which inspired the development of a robust clock-synchronization algorithm [35]. This algorithm was shown not only to tolerate up to one-third of its agents producing arbitrary or malicious pulses while the remaining agents stayed synchronized, but also to recover from arbitrary initial states, achieving synchronous pulse firing among all agents over time [35]. The latter property is known as self-stabilization [36], a particularly strong form of fault tolerance. It guarantees that even in the presence of a catastrophic event corrupting all agent states, the system can autonomously recover to normal operation without external intervention. This contrasts with non-self-stabilizing fault-tolerance, which guarantees that an algorithm can withstand a certain degree of faults. However, if the tolerated amount of faults is surpassed, the system remains faulty until it is reset to a valid configuration. Combining self-stabilization with fault tolerance therefore yields highly robust systems.

3 Optimization by Agents

In mathematical optimization, the goal is to find an optimal value $s^\star$ within a domain of possible values S, such that an objective (or loss/cost) function $f\colon S \to \mathbb{R}$ attains its minimum at $s^\star$. Analogously, a fitness function f is to be maximized. Since maximizing a fitness function is equivalent to minimizing $-f$, we will discuss only the case of a loss function.

There exists a large body of work on algorithms for finding the optimal $s^\star$. Algorithms may exploit specific properties of the set S and the function f. A key distinction among general algorithms is whether they are designed for differentiable loss functions or for general, possibly non-differentiable, loss functions. Before discussing optimization methods for loss functions in Sections 3.2 and 3.3, we begin with distributed computing problems that can be viewed as

optimization problems for a restricted class of loss functions in Section 3.1.

3.1 Optimization of a Graph

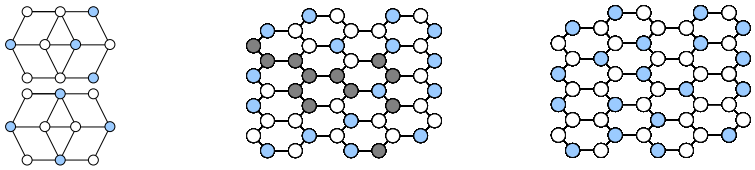
We begin our discussion of optimization by presenting two examples of distributed optimization observed in biological systems. Both examples involve identifying optimal substructures within graphs, such as minimal spanning trees, shortest paths, or maximal independent sets. Graph optimization problems have been extensively studied in computer science, in both centralized [37] and distributed [38] settings. Interestingly, both biological examples employ distributed optimization strategies, where each node updates its state based solely on information from its neighborhood, ultimately achieving an optimal solution. We emphasize that biological systems are particularly well-suited to inspire distributed strategies, as their biophysical constraints naturally limit them to rely on local information.

Maximal independent sets. Given an undirected graph $G = (V, E)$, with set of nodes V and set of edges E, a *maximal independent set* (MIS) is a subset of the graph's nodes such that **(i)** no two nodes that are neighbors in the graph are in the MIS, and **(ii)** the MIS is maximal with respect to set inclusion, i.e., no node can be added to the set without violating the first property (Figure 3a). Finding an MIS can be formulated as finding a set $S \in V$ of nodes that minimizes the following loss function:

$$f(S) = \#\{(u, v) \in S \times S \mid \{u, v\} \in E\} + \#\{v \in V \setminus S \mid \forall u \in S \colon \{u, v\} \notin E\}$$

The set S satisfies property (i) if and only if the number of edges between nodes in S, i.e., the first term in the definition of f, is zero. Furthermore, a set satisfying property (i) is maximal, i.e., satisfies property (ii), if and only if every node outside S has a neighbor in S, i.e., if the second term in the definition of f is zero. An MIS can be computed by a centralized algorithm, but distributed solutions—where graph nodes act as agents and communicate over the graph's links with neighboring agents—have been proposed [39, 40]. An example of an application of distributed MIS computation is the avoidance of collisions between neighboring radio transmitters.

Afek, Alon, Barad, Hornstein, Barkai, Bar-Joseph, and Ziv [41] showed that cells from otherwise equivalent precursors in the fly are selected to form sensory bristles, with the selected cells constituting an MIS. This observation inspired a new distributed algorithm (Figure 3b-c, Supplementary Video S5) that solves the MIS problem using only single-bit messages between cells. Agents repeatedly broadcast a single-bit message to their neighbors with a probability that increases over time, or remain silent during a round. An agent that sends a message and does not receive any other messages in the same round joins the MIS; otherwise, it does not join and continues the process. Terminating the algorithm after a sufficiently large number of rounds has been shown to generate an MIS with high probability [41], and to always ensure that no two neighboring nodes join the set.



(a) Two MIS. (b) MIS at round 9. (c) MIS at final round 19.

Figure 3: Computing an MIS on a graph: **(a)** Two maximal independent sets (MIS) for the same graph. Nodes belonging to the MIS are in blue. **(b-c)** Execution of the distributed MIS algorithm from [41]. Undecided nodes in gray, nodes in the MIS in blue, nodes not in the MIS in white.

Shortest paths. Another example of a distributed computation on a graph by a biological system is the formation of transport networks by the slime mold *Physarum polycephalum* [42]. In wet-lab experiments, *P. polycephalum* has demonstrated the ability to find shortest paths in mazes [42] and minimum-risk paths [43]. The dynamics of *P. polycephalum* can be modeled as an electrical network with time-varying resistors that respond to the electrical current through their edges, described by ODEs [44]. At each step of the model simulation, the electrical network is updated by increasing or decreasing a resistance (that can be viewed as an agent) depending on its current (that can be seen as communication with neighboring agents). It has been formally proven that the electrical network, and thus the abstract slime network, converges towards the shortest path between the two nodes with an electrical input and output current [44, 45]. This model led to many *Physarum*-inspired algorithms used to address real-world problems, e.g., traffic assignment problems [46], the design of integrated circuits [47], or the design of efficient multi-commodity flow networks [48]. More generally, *P. polycephalum*'s model has been extended to address broader classes of optimization problems: linear programs with a non-negative cost vector [49, 50], and positive semi-definite programs [51]. Similar problem-solving capabilities have also been observed in ants [52, 53].

Interestingly, results obtained on the computational side for the abstract *Physarum*-inspired model raise further research questions on the biological side, such as gaining new insights into tolerance to noise in biological systems, informed by the study of noise tolerance in the simplified computational model.

3.2 Direct Search: Exploring Solutions via Repeated Loss Evaluations

In the general case, where the function is not necessarily differentiable, so-called direct search methods are commonly employed [54, 55, 56]. These methods do not rely on derivatives but instead evaluate the function at different points in S. Clearly, though, the distinction is rather technical, as function evaluations may, in some cases, be used to numerically approximate the gradient ∇f.

Common examples of direct search methods include genetic algorithms [55], differential evolution [56], particle swarm optimization [57], Nelder–Mead [58], and Hooke–Jeeves [54], among many others.

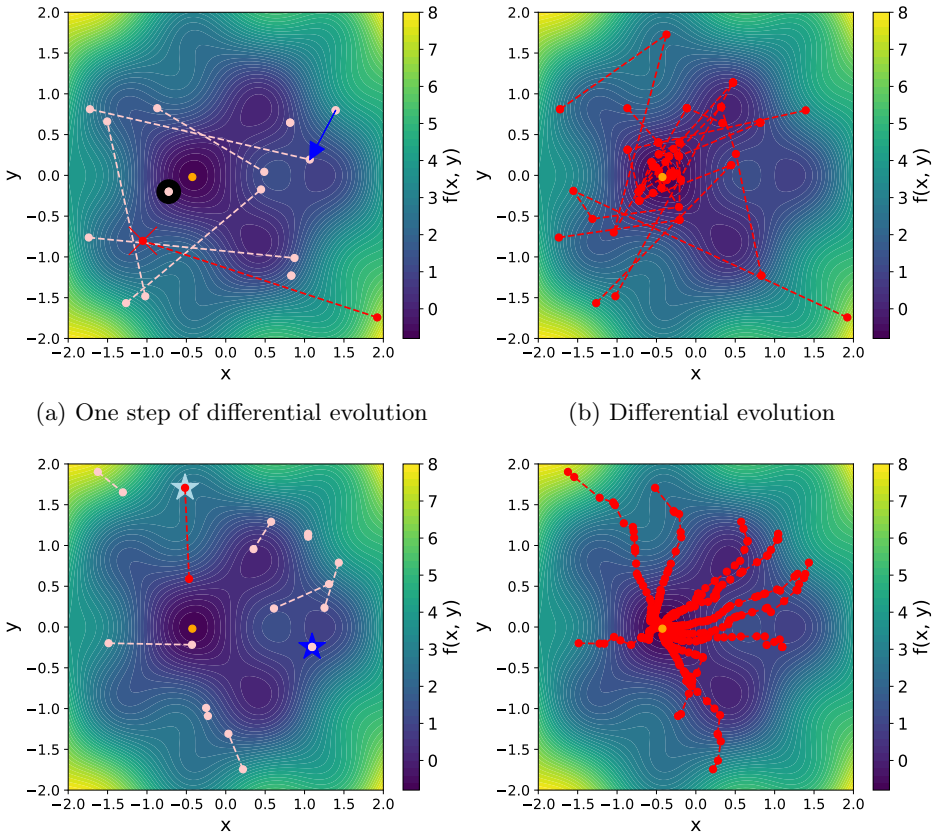
Genetic algorithms. Genetic algorithms [55] resemble an abstract microbiological setting. Repeatedly, solution candidates are evaluated for fitness. Based on these evaluations, a subset of candidates is selected to generate a new population through mutation (changes derived from a single candidate) and recombination (changes derived from multiple candidates). The selection, mutation, and recombination processes are stochastic, with mutation and recombination typically defined on the encoding of candidate solutions, inspired by mutations and recombination in biological genomes.

Again, this provides a clear example of an intentionally simplified yet successful model, where closeness to the biological counterpart is not intended: selection, mutation, and recombination fall short of replicating the corresponding biological processes.

Differential evolution. Differential evolution [56], closely related to genetic algorithms, operates on candidates in $\mathbb{R}^n$, with recombination defined directly on these candidates. While several variants of the algorithm exist, we focus on a basic version. The algorithm repeatedly, for each candidate, generates a new candidate by combining it with three other randomly selected candidates: Changes are applied to a randomly selected subset of candidate components, obtained by adding $F \cdot (s_c - s_b)$ to a randomly chosen candidate s_a, where $F \in \mathbb{R}$ and s_a, s_b, s_c are three distinct randomly chosen candidates (Figure 4a). If the new candidate shows an improvement in fitness, the old candidate is replaced by the new one. Figures 4a-b and Supplementary Video S6 illustrate differential evolution in action on an example with candidates in $\mathbb{R}^2$. Candidate updates can be performed in parallel during each iteration, making the algorithm efficient for distributed computation.

Particle swarm optimization. Particle swarm optimization (PSO) [57] is an optimization strategy inspired by the collective search behavior of biological entities in spatial environments—such as fish, ants, birds, and other cooperative communities that communicate the locations of food sources. PSO abstracts this communication as the sharing of an optimum among directly connected entities. Starting from an initial population of particles with associated velocities, the algorithm iteratively updates each particle's velocity as a weighted sum of three components: **(i)** its previous velocity (weight w), **(ii)** the direction toward its personal best position (weight sampled uniformly in $[0, c_1]$), and **(iii)** the direction toward the best position found by its neighbors (weight sampled uniformly in $[0, c_2]$). The particle's position is then updated based on the new velocity. Several variants of PSO have been proposed.

Figures 4c-d and Supplementary Video S7 illustrate PSO in action, with Figure 4c showing a single update step and Figure 4d showing the trajectories



(a) One step of differential evolution

(b) Differential evolution

(c) One step of particle swarm optimization

(d) Particle swarm optimization

Figure 4: Optimizing candidate solutions for the function f (see Equation (2)). The global optimum is shown in orange. **(a).** A step in differential evolution with population size 10. The trajectory of one candidate (in red) is shown for the case $F = 1$ and all components of a candidate being updated ($R = 1$). Other candidates are shown in white. The blue arrow marks $F \cdot (s_c - s_b)$ which is added to s_a (in black), resulting in the updated position of the candidate (red cross). The solution is accepted since its loss is less than at the initial position of the candidate. **(b).** Differential evolution: Trajectories of all 10 population members are shown in red ($F = 0.7$ and $R = 0.9$). **(c.)** A step in particle swarm optimization with 10 particles and a fully connected communication graph ($w = 1$, $c_1 = 1.5$, $c_2 = 0.6$). The update of a single particle (in red) is shown while other particles are shown in white. The update results in a move in the combined direction of the current velocity, the direction to the own optimum (light blue star), and the swarm-wide optimum (blue star). **(d).** Particle swarm optimization: Trajectories of all 10 particles are shown in red ($w = 0.05$ and $c_1 = c_2 = 0.15$).

of all 10 particles.

In practice, communication between entities is modeled through communication graphs, that is, graphs with particles as nodes and having an edge from particle a to particle b if a can send messages to b. Typical communication graphs are either fully-connected graphs or rings. Particle updates can also be performed in parallel during each iteration, with a sparsely connected communication graph, such as a ring, reducing communication overhead.

Finally, we emphasize that PSO can be viewed as a generalization of the simpler agreement algorithms discussed in Section 1, which becomes apparent by setting $c_1 = 0$ and using a constant loss function.

3.3 Gradient-based Optimization

Many loss functions f are, or can be chosen to be, differentiable (almost everywhere). While this may seem like a minor assumption, it enables optimization techniques that can scale to millions of parameters—domains where the previous methods clearly struggle.

For a differentiable function f, its derivatives can be used to iteratively update a candidate solution $s \in S$. One approach is to move the candidate s in the opposite direction of the gradient ∇f at s. This corresponds to moving the candidate along the direction of steepest descent of a linear approximation of f. While the most fundamental update schemes apply this rule directly, more advanced ones leverage additional properties, such as accounting for the previous update direction, e.g., incorporating concepts analogous to the momentum of a mass in a physical system [59]. These approaches are widely used in the training of artificial *neural networks* (NNs), as discussed in the following.

Perceptron model and neural networks. Models of biological neurons, as discussed in Section 4, are typically described using non-linear differential equations involving chemical signal concentrations and electrical input and output currents. For reasons of computational complexity, the concept of a biological neuron is abstracted into an artificial neuron, without kinetics.

A standard artificial neuron, such as the Perceptron [60], can be described as taking a fixed input current (or simply a value), typically computed as a weighted sum of multiple input currents (values) in_i, and applying a non-linear activation function g to determine the (fixed) output current (value):

$$\mathrm{out} = g\left(\sum_i w_i \cdot \mathrm{in}_i\right) \quad . \tag{1}$$

For a fixed g, The neuron is parametrized by its input weights w_i that determine its input-output behavior. In the simplest case, g is the Heaviside step function: $g(x) = 1$ for $x \geq 0$ and $g(x) = 0$ otherwise. However, functions with more favorable properties for training networks of neurons—such as facilitating the parametrization of weights and function parameters—are typically used. Common examples of non-linear activation functions include

sigmoid-type functions, ReLU, i.e., $g(x) = \max(0, x)$, and Leaky ReLU, i.e., $g(x) = \max(0, x) + \alpha \cdot \min(0, x)$, which preserve gradient information from inputs to output, while Leaky ReLU also allows a small gradient for negative inputs. Meanwhile generalizations of the input-output behavior as stated in Equation (1) to other differentiable functions are commonly used. Examples are products of inputs and soft-max.

In contrast to these current-based neurons, spiking neurons, which form the basis of spiking neural networks (SNNs), are closer mimicking biological neurons, by not abstracting from the dynamics over time. Spiking neurons communicate via timed events or spikes rather than propagating a single continuous activation level [61]

Building on the concept of a single neuron, neural networks (NNs) are formed by connecting multiple neurons, with current networks reaching having billions of learnable parameters.

Feedforward neural networks (FNNs) are the simplest type of network, where neurons are organized in layers and information flows strictly from the input layer through hidden layers to the output layer. Computationally, perceptrons in a layer are typically collapsed into vectors and higher-dimensional tensors, and a linear transformation followed by a non-linear activation function is applied to obtain the tensor of the next layer.

Recurrent neural networks (RNNs) extend FNNs by introducing cycles that allow the network to maintain a hidden state, enabling the processing of sequential or time-dependent data [62]. Since RNNs can also be viewed as repeatedly applying the same FNN to a sequential input and the hidden state, it becomes clear that gradients may vanish (zero-out) with each iterative application. A special class of RNNs, long short-term memory (LSTM) networks, is designed to mitigate this problem through memory cells with gates that regulate which information is stored, updated, or output at each iteration [63].

Convolutional neural networks (CNNs) [64] are specialized for processing data with spatial or temporal structure, such as images or signals. Instead of learning independent weights for each connection, CNNs apply a set of repetitive kernels (filters) across the input, sharing parameters across positions to exploit local correlations and reduce the number of trainable parameters.

Transformers [65] rely on encoders, decoders, and attention mechanisms to directly model dependencies between elements (tokens) in a sequence of fixed, but potentially very long, length, enabling the capture of long-range relationships. They have proven particularly effective for natural language processing and code generation via large language models (LLMs), among many other applications.

Graph neural networks (GNNs) [66] generalize neural network architectures to graph-structured data. Unlike Transformers, which operate on fully connected sequences, where each token can be in relation to each other token, GNNs operate on nodes connected according to a graph structure. The value of each node is iteratively updated by applying neural networks to the features of its neighboring nodes, allowing the network to capture the relational structure of the graph.

Training neural networks. The training, i.e., parametrization, of artificial neural networks remains an active research area, with ongoing discussions on the parallels between learning in biological and artificial systems [67, 68, 69]. Typically, backpropagation, a gradient-based optimization technique, is used to train artificial NNs. In this approach, the output of an artificial NN is determined for an input x by evaluating the neurons' functions in a forward pass, storing additional information that is then used in a backward pass to determine the gradient of the network for input x in a backward-pass.

Consider the simplest case of training an NN to predict target outputs $y_{\text{target}}(x)$ for given inputs x. Let $y_{\text{model}}(x;\theta)$ denote the outputs of the NN with parameters $\theta \in S$ for x. The loss function $f(\theta;x)$ is given by:

$$f(\theta;x) = \|y_{\text{model}}(x;\theta) - y_{\text{target}}(x)\|$$

where $\|\cdot\|$ is a properly chosen norm. The parameters $\theta \in S$ are typically optimized either directly via gradient descent on $f(\theta;x)$, or by combining several inputs into a batch B and optimizing the combined loss, e.g., $\frac{1}{|B|}\sum_{x\in B} f(\theta;x)$. The latter approach underlies stochastic gradient descent (SGD). In addition to introducing stochasticity through randomly formed batches, batching enables parallel computation of gradients for all inputs in the batch. The resulting gradients are then averaged to update θ.

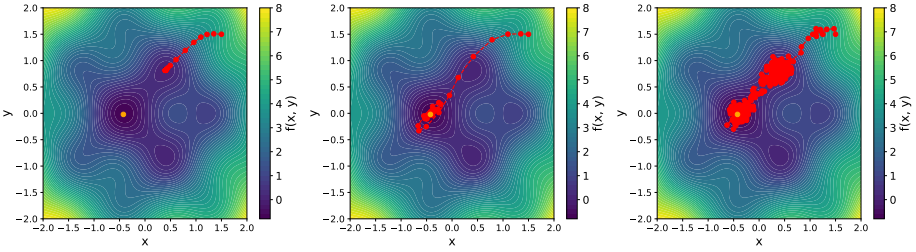
For the following, consider the following objective function f defined over the domain $S = \mathbb{R}^2$.

$$f(x,y) = \sin(\pi x)\cos(\pi y) + x^2 + y^2 \tag{2}$$

This function exhibits multiple local minima and a unique global minimum. Figure 5 illustrates gradient descent trajectories from a given initial state. With the chosen parameters, standard gradient descent tends to get trapped in local minima (Figure 5a, Supplementary Video S8). Introducing momentum (Figure 5b, Supplementary Video S9) or noise components (Figure 5c, Supplementary Video S10) facilitates escape from local minima and enables convergence to the global minimum.

Reinforcement learning. In the context of artificial NNs, another similarity between computational algorithms and biological systems can be observed. *Reinforcement learning* (RL) algorithms closely resemble presumed learning strategies of biological entities. While early algorithms like Q-learning [70] did not make use of NNs, many state-of-the-art versions do.

RL algorithms are typically formulated within the framework of *Markov decision processes* (MDPs). An MDP is defined by **(i)** a set of states representing both the agent and the environment, **(ii)** a set of possible actions available to the agent, **(iii)** a probabilistic transition function mapping each state–action pair to a distribution over next states, and **(iv)** a reward function assigning immediate rewards to states or state-action pairs. The agent typically cannot observe the complete state and must choose actions based only on the observable components. The agent's objective is to maximize the expected long-term



(a) Gradient descent. (b) with momentum. (c) with a noise component.

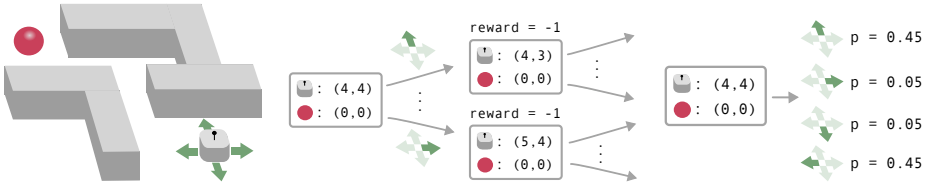
Figure 5: Maps of the values of f, described by Equation (2), for $x, y \in [-2, 2]$. The global optimum is shown in orange. Candidate optimal state trajectories are shown in red. These trajectories result from (**a**) standard gradient descent, (**b**) gradient descent with momentum, and (**c**) gradient descent with a noise component, where every second step is done in a Gaussian-distributed random direction instead of following the gradient.

reward, typically computed as the cumulative (e.g., summed) sequence of immediate rewards. An illustrative example is shown in Figure 6, where a robot navigates a simple grid-world environment to locate a red ball (Figure 6a). The corresponding MDP and the robot's strategy are depicted in Figures 6b and 6c, respectively.

A widely adopted algorithm for learning such strategies is Q-learning [70]. In this approach, the Q-value of a state-action pair represents the estimated total cumulative reward obtained by taking that action in the given state and following the optimal policy thereafter. Q-values are iteratively updated during learning: after taking an action in a state and observing the resulting reward and next state, the Q-value is adjusted toward the observed immediate reward plus the (discounted) maximum Q-value of the next state, as currently estimated from previous learning iterations.

Modern RL algorithms employ neural networks to estimate the expected reward of an action (value-based RL, e.g., DQN [71]), to directly represent the policy (policy-based RL, e.g., REINFORCE [72]), or to implement hybrid approaches such as actor-critic RL (e.g., A3C [73]). Gradient-based optimization techniques applied to appropriately defined loss functions are used to optimize these NNs.

Beyond the use of neural networks, RL algorithms in computer science incorporate a concept that is particularly relevant for comparison with learning in biological organisms: the trade-off between exploration and exploitation. This concept is closely related to the hypothesized learning mechanisms of biological agents, which adapt their behavior to maximize long-term rewards. For both biological and algorithmic agents, extreme strategies are generally ineffective: solely exploiting previously optimal actions (exploitation-only) or exclusively exploring previously untried scenarios (exploration-only). In Q-learning, where the expected cumulative reward of each action is iteratively estimated during



(a) A robot (agent) in a grid world environment.

(b) Corresponding Markov decision process.

(c) A probabilistic strategy.

Figure 6: Example of a learning agent interacting with an environment: a robot must navigate a grid world with obstacles to find a red ball (**a**). The sets of the robot's actions are modeled by a *Markov decision process* (MDP). States are coordinates, and actions movements of the robot and the ball (**b**). The probability distribution is trivial as the environment is deterministic. The agent receives an immediate reward of -1 per step and 10 upon reaching the ball. A probabilistic strategy mapping states to action probability distributions is given in **c**.

learning, a common strategy is to select a random action with a small probability $\varepsilon > 0$ and, with complementary probability $1 - \varepsilon$, choose the action that currently maximizes the expected long-term reward.

In general, the combination of exploration and exploitation can also mitigate a potential problem of purely gradient-based optimization: the attraction to local minima and saddle points. A biological example of combining gradient-based search with exploration is the chemotaxis of bacteria such as *Escherichia coli*. By alternating between runs and tumbles that randomly change the swimming direction, the cell adapts tumble frequency depending on how well the run direction is aligned with the gradient of a chemical attractant [74]. We hypothesize that studying analogous strategies in computer science could provide insights when compared to the commonly used stochastic gradient descent.

Of particular interest in distributed computing is multi-agent reinforcement learning (MARL), where multiple agents learn to act in a shared environment. The agents may compete, cooperate, or engage in mixed strategies to achieve a common goal. Example applications include multi-player video games, modeling social behavior, auctions, and coordinating cooperative drones, among others A major challenge in MARL is the combinatorial growth of the joint action space as the number of agents increases. For instance, the output layer of a joint policy network would scale exponentially with the number of agents. While one could train agents individually, this approach does not promote cooperation. One strategy to mitigate this problem is to exploit symmetries among agents: if the agents are interchangeable, the learning algorithm can be chosen to be *equivariant* by design, i.e., symmetric with respect to permutations of agent states, without the NN having to learn this symmetry. Architectures such as DeepSets [75], DPN and HPN [76], and PEDQN [77] are based on this approach, and leverages equivariance to efficiently learn in multi-agent environments.

4 Competing Neuron Models

In this section, we compare a mechanistic biological neuron model with the abstract neuron model used in artificial neural networks introduced in Section 3.3, emphasizing their conceptual and functional differences.

Mammalian brains contain large populations of interacting neurons whose collective activity enables complex behavior. A widely used mechanistic description of a single neuron is the Hodgkin–Huxley (HH) model [78]. It describes how the membrane potential changes over time in response to an input current density. The model is expressed as a system of non-linear *ordinary differential equations* (ODEs) that describe the charging and discharging of the membrane capacitance. Charging happens via an input current density $I(t)$ that results in an altered membrane potential $V(t)$, following the ODE

$$C \cdot \frac{dV(t)}{dt} = I(t) - I_{\mathrm{K}}(V(t)) - I_{\mathrm{Na}}(V(t)) - I_{\mathrm{L}}(V(t)) \tag{3}$$

where C is the capacitance. The non-linear discharging current densities of the sodium (Na), potassium (K), and leak (L) channels are described by

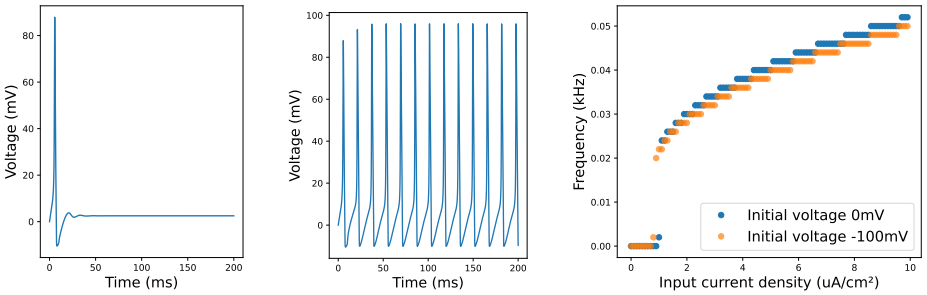
$$\begin{aligned} I_{\mathrm{Na}}(V) &= g_{\mathrm{Na}} \cdot m^3(t) \cdot h(t) \cdot (V - V_{\mathrm{Na}}) \\ I_{\mathrm{K}}(V) &= g_{\mathrm{K}} \cdot n^4(t) \cdot h(t) \cdot (V - V_{\mathrm{K}}) \\ I_{\mathrm{L}}(V) &= g_{\mathrm{L}} \cdot (V - V_{\mathrm{L}}) \end{aligned}$$

where the dynamics of $m(t)$, $n(t)$, and $h(t)$ are defined by

$$\begin{aligned} \frac{dm(t)}{dt} &= \alpha_m(V(t)) \cdot (1 - m(t)) - \beta_m(V(t)) \cdot m(t) \\ \frac{dn(t)}{dt} &= \alpha_n(V(t)) \cdot (1 - n(t)) - \beta_n(V(t)) \cdot n(t) \\ \frac{dh(t)}{dt} &= \alpha_h(V(t)) \cdot (1 - h(t)) - \beta_h(V(t)) \cdot h(t) \end{aligned}$$

and α and β are scaled and shifted exponential functions of V. We refer the reader to the original paper [78] for the choice of these functions and the parametrization of the involved constants like the capacitance C and threshold potentials V_{Na}, V_{K}, and V_{L}. A Python implementation is available at `https://github.com/BioDisCo/dc_bio`.

The model shows interesting behavior of its output (the membrane potential) in presence of a step function as its input current density. In accordance with the Perceptron-like model, some threshold-like behavior can be observed. If the amplitude of the step is below a certain threshold, the output is a single short pulse (Figure 7a). Increasing the amplitude of the input above a threshold leads to a train of pulses at the output (Figure 7b). It has been noted [79] that, above the threshold, the frequency of the pulse train increases logarithmically with the



(a) Input height below threshold.

(b) Input height above threshold.

(c) Frequency of the pulse-trains as a function of the height of the input current density for two initial states.

Figure 7: ODE simulations of a Hodgkin–Huxley model. Blue lines are simulations for an initial state set to 0 mV, orange line is for an initial state set to −100 mV. Input current density is a Heaviside step signal of varying height at 10 ms. The resulting membrane potential of the neuron is shown for an input below the threshold (**a**) and above the threshold (**b**). Input height above threshold results in a pulse-train. The frequency of the pulse train is a function of the height of the input current density and initial state (**c**).

input step amplitude (Figure 7c). This reveals an important contrast with common non-linearities in artificial neurons, which typically rely on piecewise-linear, sigmoid, or exponential functions rather than threshold-logarithmic relations.

One also observes a second fundamental difference: care has to be taken with the observation and definition of a threshold. While in artificial neurons sharp thresholds are often used (e.g., in ReLu) and can be explicitly shifted with a so-called bias of the neuron, the threshold in the Hodgkin–Huxley model, and in general, the input–output behavior depends on the initial state of the system. While for our purposes of a simplified demonstration the initial state was set to all 0, a biological neuron will clearly not be reset to this null state after each activation. Consequently, the effective threshold depends on the neuron's initial state (Figure 7c). We would like to stress that—much like the initially mentioned airplane design that currently does not follow the mechanics of a bird—the mismatch between the artificial model and the biological counterpart should not be seen as a shortcoming. It may very well be that abstracting the model into a state-less simplified model enables today's large artificial neural networks.

Several models that are more abstract than the classical Hodgkin–Huxley model, but still mechanistic, have been proposed. They differ in the aspects of dynamic behavior that that capture or simplify. The FitzHugh–Nagumo (FN) model [80, 81] is a simplification of the Hodgkin–Huxley model from four state variables (V, m, n, and h) to two (x and y) while still capturing the properties of producing a pulse train oscillation upon a sufficiently large input current.

This simplification facilitates analysis and reduces computational cost, which is particularly important for large neuronal networks. In this model, z represents the input current and x the output potential. The dynamics are specified as

$$\frac{dx(t)}{dt} = c \cdot \big(x(t) + y(t) - x^3(t)/3 + z(t)\big)$$
$$\frac{dy(t)}{dt} = -\frac{1}{c} \cdot (x(t) - a + b \cdot y(t))$$

with constants a, b, and c.

Even more abstract are integrate-and-fire models, in which a neuron integrates incoming currents, as in Equation (3) but setting the currents $I_{\mathrm{K}} = I_{\mathrm{Na}} = I_{\mathrm{L}} = 0$, until a threshold potential is reached, a spike is triggered, and the potential is reset. Variants include a leaky term, by adding a resistance in parallel to the capacitance, or account for a refractory period after a spike that imposes a maximum firing frequency.

At the other end of the spectrum, more complex models that include stochastic processes in the nervous system [82] have been studied, e.g., by adding noise to the Hodgkin–Huxley model [83]. Together, these models illustrate a continuum between biological realism and abstraction—providing a mechanistic contrast to the stateless, algebraic neurons of artificial networks discussed earlier.

In the next section, we present examples where proximity to a biological model is essential. The examples describe distributed systems intended for implementation in biology. Although simplifications are needed to make such models analyzable, they must still include key properties of the biological system to remain relevant. In the case of an example of a distributed algorithm designed for engineered bacteria that is detailed, such key properties are the stochasticity of the system and bacterial growth during algorithm execution.

5 Biology Inspired by Distributed Computing

Viewing cells as autonomously computing devices reveals a multitude of natural and engineered distributed systems in biology. Indeed, it may be challenging to identify biological systems that are centralized or composed of single, non-interacting cells.

5.1 Engineering Biological Distributed Systems

Many biological processes, such as pattern formation in spatially distributed cells, are inherently distributed problems. Examples include the engineering of spatial patterns in cellular populations [84] and the design of bacterial cells whose response depends on the cell's spatial localization, to implement distinct digital gates [85].

Similarly, clock synchronization among engineered bacteria is an inherently distributed task and has been used to coordinate the timed release of therapeutic compounds, with therapeutic applications [86].

Distributed circuit architectures have also emerged out of resource limitations, including the limited availability of orthogonal wiring molecules within a single cell and the metabolic burden imposed by large genetic circuits. While initial work in synthetic biology focused on single-cell genetic circuits [87, 88, 89], subsequent studies extended these ideas to multicellular systems that distribute computation across communicating populations. Examples include distributed computation in engineered yeast networks using pheromone-based signaling [90], robust multicellular computing in *E. coli* based on genetically encoded NOR gates and quorum-sensing communication [91]. We refer the interested reader to comprehensive surveys on multicellular computation [92, 93].

Communication is a central component of distributed biological computation. The most widely used mechanism relies on diffusible quorum-sensing molecules. Systematic frameworks for circuits employing quorum-sensing communication have been developed to implement Boolean functions in a distributed manner [94]. Recent studies have demonstrated scalable implementations, achieving large distributed circuits [95]. Alternative communication modalities have also been explored. Engineered bacteriophages have been demonstrated to communicate digital information between bacteria [96], and was used in constructing distributed genetic circuits in *E. coli* [97, 98]. Further, plasmid exchange has been exploited as a mechanism for targeted communication between bacteria [99].

Closely related to multicellular systems are biochemical networks composed of interacting molecular species that react, form complexes, or catalyze reactions. A prominent example is computation through interacting DNA molecules. DNA has been used for programmed self-assembly [100], molecular computation [101], distributed DNA-based circuits that enable multichannel molecular communication between populations of non-lipid microcapsules [102], and biomedical applications such as targeted drug delivery [103]. Inspired by reconfigurable electronic hardware such as Field-Programmable Gate Arrays (FPGAs), scaling to larger circuits with DNA registers has been demonstrated feasible [104]. Such circuits also make use of concepts from asynchronous circuit design such as dual-rail logic gates.

Recent developments in microbial engineering have produced tools to engineer microbial communities [105], screening for inter-population interactions that yield stable steady-state co-cultures. Other approaches explore evolutionary design strategies to obtain desired system-level behaviors without explicitly engineering individual cells [106].

5.2 Agreement among Bacteria

In this section, we present a distributed design of a bacterial system that solves a classical problem from distributed computing: asymptotic agreement. A distinctive feature of this design is that it can be modeled and analyzed using tools analogous to those employed in theoretical computer science for distributed algorithms. In particular, explicit mathematical guarantees have been proven for its dynamical behavior [107, 108].

The distributed computing community has introduced several models that are intended to capture aspects of biological systems, including gossiping [109], population protocols [110], stone-age computing [111], the beeping model [112], and certain shared-memory models [113]. For the analysis of bacterial populations the formalism of chemical reaction networks (CRNs) provides a particularly natural and accurate framework.

Chemical reaction networks. Originally developed to formalize chemical kinetics and thermodynamic reasoning, CRNs [114, 115] have also been extensively studied within theoretical computer science and are of independent computational interest [116, 117, 118]. A CRN is a mathematical model to describe and analyze chemical reaction kinetics. Formally, a CRN is defined by a finite set of species $A, B, \ldots$, a set of reactions operating on these species, and an initial state specifying the initial abundance of each species. Each reaction takes the general form

$$r_A A + r_B B + \ \ldots \quad \longrightarrow \quad p_A A + p_B B + \ \ldots$$

indicating that r_A of the reactant A, r_B of the reactant B, etc., react to form p_A of the product A, p_B of the product B, etc. Each reaction is assigned a rate, often of the form of mass-action kinetics. CRN dynamics can be modeled either stochastically—as a continuous-time Markov chain—or deterministically, by a system of ODEs. In the deterministic setting, the reaction rate under mass-action kinetics is proportional to the product of the reactant concentrations, i.e., of the form $\gamma \cdot \left(\frac{A(t)}{v}\right)^{r_A} \cdot \left(\frac{B(t)}{v}\right)^{r_B} \cdots$, where $\gamma > 0$ is the rate constant, $v > 0$ the system volume, and $S(t)$ the abundance of species S at time t. Alternative formulations, consider $S(t)$ to be concentrations, removing the division by v. For stochastic semantics, mass-action rates are proportional to the number of distinct combinations in which the reactant molecules can be selected. For instance, a reaction with two identical reactants, $A + A \longrightarrow \ldots$, occurs with a rate (and has a propensity) proportional to $\binom{A(t)}{2} = A(t) \cdot (A(t) - 1)/2$, reflecting the number of unordered pairs of A molecules. In general, the propensity is proportional to $\binom{A(t)}{r_A} \cdot \binom{B(t)}{r_B} \cdots$.

While originally developed for chemical reactions, CRNs have also been employed to model and parameterize a wide range of biological processes, including the dynamics of natural [119] and synthetic [120] genetic networks. Numerous software tools have been developed to efficiently specify and simulate CRNs [121, 122, 123, 124, 125].

For distributed computing, CRNs are appealing because they represent one of the computationally simplest models of interacting agents, while still being mathematically analyzable despite their inherent complexity. They model single chemical species as agents that interact with other agents, leading to state transitions (e.g., $A + B \longrightarrow A + A$) or the creation of new agents (e.g., $A + B \longrightarrow A + A + B$). Their simplicity has motivated extensive studies of their computational expressiveness, including in the even more constrained setting of population protocols with uniform reaction rates and constant population

sizes, which already pose interesting challenges [4]. In microbiological contexts, CRNs have been applied to the design of self-assembling DNA tiles [101], as well as to the mathematical analysis and implementation of distributed algorithms that achieve agreement in bacterial populations [107, 108] or detect specific compounds of interest [126].

Figures 8a–b illustrate simulations of a simple CRN (ABC) comprising the reactions

$$A + B \longrightarrow A + C \quad [0.1] \tag{4}$$

$$A \longrightarrow \emptyset \quad [0.1] \tag{5}$$

with initial counts $A = B = 10$ and $C = 0$, system volume $v = 1$, and mass-action kinetics using the rate constants shown in brackets. Figure 8a presents the deterministic simulation, whereas Figure 8b displays stochastic simulation results for the same network. A pronounced variability across stochastic runs is clearly visible.

An Algorithm for Bacteria. In [107, 108], CRNs where used to study if bacteria can be used to solve the majority consensus problem, a particular instance of asymptotic agreement: starting from a system of agents with initial values (A or B), the system converges towards a system with all agents having the value that was initially in majority. In the bacterial system, agents are bacteria of type A and B that grow on a common resource R and that are engineered to compete with each other's type, making one of them die upon interaction.

Figures 8c-d show simulations of the respective *MutualAnnihilation* CRN similar to those analyzed in [107, 108] with reactions

$$A + B \longrightarrow A \quad [0.1] \tag{6}$$

$$A + B \longrightarrow B \quad [0.1] \tag{7}$$

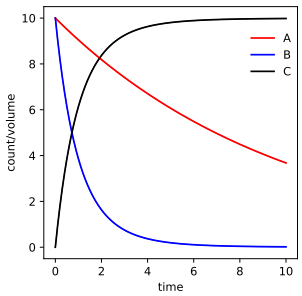
$$A + R \longrightarrow A + A \quad [0.01] \tag{8}$$

$$B + R \longrightarrow B + B \quad [0.01] \tag{9}$$

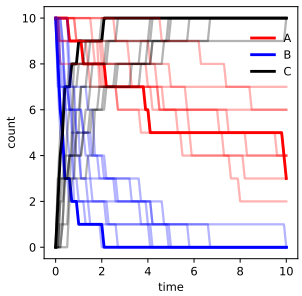
initial counts $A = 12$, $B = 8$, and $R = 100$, volume $v = 1$, and mass-action kinetics with reaction rate constants noted in brackets. Figure 8c shows a deterministic simulation and Figure 8d stochastic simulations for the CRN. While in the deterministic simulation, the initial majority (A) always wins, the stochastic simulation captures uncertainties in the interaction among the competing bacteria, with a high—but less than 1—probability that A wins.

The *MutualAnnihilation* CRN is particularly interesting because it models competition among growing species, such as bacteria, while accounting for varying population sizes and the stochastic fluctuations expected in real-world implementations.

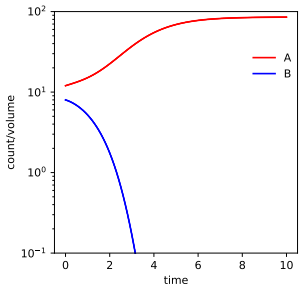
In fact, the runs in Figure 8d exemplify a strong amplification effect. As shown in [107, 108], the bacterium initially in the majority has a higher probability of winning against its competitor, with this probability increasing faster



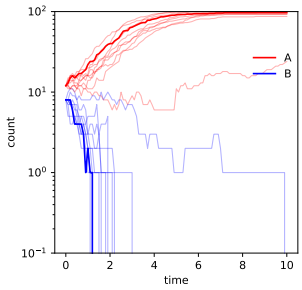
(a) Deterministic simulation of the *ABC* CRN.



(b) Stochastic simulation of the *ABC* CRN.



(c) Deterministic simulation of the *MutualAnnihilation* CRN.



(d) Stochastic simulation of the *MutualAnnihilation* CRN.

Figure 8: Simulations of two chemical reaction networks (CRNs): *ABC* described by Equations (4) and (5) (**a-b**); and *MutualAnnihilation* described by Equations (6)–(9) (**c-d**). Figures **a-c** are deterministic simulations. Figures **b-d** show 10 stochastic simulations of each CRN. A single trace is shown in solid, 9 others in light.

than linearly with the initial gap, $\max(A, B) - \min(A, B)$. For large population sizes n, an initial gap of order $\sqrt{n}$ is sufficient for the majority to prevail with high probability. In particular, this demonstrates that, for sufficiently large n, the initial majority need not exceed 51% to win with high probability.

6 Conclusion

Analogous concepts in distributed computing and biological systems have been discussed with inspirations and commonalities between the two disciplines.

We highlighted selected fundamental problems that involve multiple agents solving increasingly complex tasks, with an emphasis on reaching agreement among agents. We started from one of the most basic tasks, agreeing on a common value, to increasingly general optimization problems, where agents agree on optimal values. The examples cover aspects that were emphasized by Navlakha

and Bar-Joseph [1] as commonalities between biological and computational systems: decentralization, robustness to failures and noise, communication of information via dedicated networks, heavy (re)use of modular components, and an inherent stochasticity of the involved processes; with Section 5 discussing the reverse direction of applying distributed computing conepts to biological systems. In particular, we discussed a distributed algorithm intended for implementation bacteria that as an example for a problem that involves all five aspects.

Clearly, though, the problems and algorithms covered in this write-up are necessarily only a short overview. Other examples, like the social behavior and foraging strategies of animals like ants [127, 128] and bees [129], are promising fields for commonalities to lead to a deeper understanding of the biological system and new algorithms. We expect many more such commonalities to be discovered in the future; with synthetic biology, where algorithms and biology come particularly close, playing an interesting role on this path.

We also tried to stress the importance of models that was already pointed out by Feinerman and Korman [7], emphasizing that while in this interdisciplinary discipline repeated cross-transfer between the two domains requires an understanding of models on both sides, differences between models of analogous problems and algorithms are to be expected and probably essential for progress in both disciplines. Clearly, though, this should not be misunderstood as advocating for the theoretical study of irrelevant models or for mechanistic biological models that have been falsified; care has to be taken on the intention of a model.

While there are numerous example from bioinformatics for classical algorithms in the interdisciplinary domain of biology and computer science, we focused on distributed systems in this review. We believe that these will play an increasingly crucial role in both domains due to the limited availability of local computation because of power limitations, noise, feasible technological structure sizes, and robustness to failures. Distributed systems are a way out of these limitations: they speed up computation by parallelizing it, synchronizing only when necessary, and they often provide robustness to perturbations in the environmental conditions and faults of some of its computational entities. Further, distributed systems are useful from an engineering perspective: they allow building complex system behavior via only a few different, but often numerous in number, simple computational building blocks.

Finally, we would like to speculate on the future potential of combining distributed computing with biology. The fact that almost no biological system is centralized strongly suggests the promise of distributed architectures in a biological context. Interestingly, complex systems—such as mammals—are composed of cells that carry identical DNA, showing that it is the distributed architecture rather than a highly refined individual program that produces sophisticated collective behavior.

By comparison, engineered systems typically exhibit a lower degree of distribution and are often less robust to deviations or faults. We therefore hypothesize that, particularly in terms of robustness, there is much to be learned from biological architectures for architectures in silicon.

Beyond analyzing natural systems, synthetic biology opens a new frontier at the interface of biology and distributed computing. It enables both the engineering of novel biological distributed systems and the study of fundamental principles in natural systems through simplified or modified behaviors. In theoretical computer science, a common approach in distributed computing is to systematically alter model assumptions—such as communication reliability, delay, or process failures—to determine the exact conditions under which a problem becomes solvable or unsolvable. We think that this is a particularly interesting route to pursue in synthetic biology, as it allows one not only to engineer and manipulate cellular interactions, but also to systematically explore which types of interactions are necessary to achieve desired outcomes. By doing so, we can uncover the minimal or sufficient conditions required for phenomena such as spatial patterning, collective agreement, or balanced co-culturing in populations of cells, as well as for many other phenomena that remain to be explored.

Methods

All simulations have been performed with Python 3.12.7 using `MobsPy` [125], `networkx`, `numpy`, `random`, and `scipy`. Integration for the Hodgkin–Huxley ODE model was done with `solve_ivp` (`scipy.integrate`) and the RK45 method. For differential evolution the method `differential_evolution` (`scipy.optimize`) was used with a custom strategy for demonstration purposes of a simple variant and plotting. The graph library `networkx` was used for managing communication graphs. The butterfly graph was built according to [24, Figure 1]). Random communication graphs were generated by creating minimum weight spanning trees of weighted complete graphs, and directing them by BFS traversal starting from the root. Three trees with different roots were generated, and, with $N = 10$ being the number of agents, $\lfloor \log(N) \rfloor \cdot N = 20$ edges were added (uniform probability of an edge being in the graph). The communication sequence was obtained by sampling uniformly at random from the created graphs. All averaging and averaging alternating with flooding algorithms were executed on the same initial values (for $\mathbb{R}$ and $\mathbb{R}^2$), the same sequence of communication graphs, and the same number of rounds (6), unless stated otherwise. `MobsPy` was used to generate stochastic runs of CRNs.

All plots are generated with `matplotlib`. Code for all simulations and the creation of the figures is publicly available at `https://github.com/BioDisCo/dc_bio`.

Acknowledgments

The work was supported by the French National Research Agency (ANR) projects DREAMY (ANR-21-CE48-0003) and COSTXPRESS (ANR-23-CE45-0013), as well as the SAIF project, funded by the "France 2030" government investment plan managed by ANR, under the reference ANR-23-PEIA-0006.

References

[1] Saket Navlakha and Ziv Bar-Joseph. Algorithms in nature: the convergence of systems biology and computational thinking. *Molecular Systems Biology*, 7(1):546, January 2011. Publisher: John Wiley & Sons, Ltd.

[2] Saket Navlakha and Ziv Bar-Joseph. Distributed information processing in biological and computational systems. *Commun. ACM*, 58(1):94–102, December 2014.

[3] Bogdan S Chlebus, Leszek Gasieniec, Alan Gibbons, Andrzej Pelc, and Wojciech Rytter. Deterministic broadcasting in ad hoc radio networks. *Distributed computing*, 15(1):27–38, 2002.

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

[5] Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *International Symposium on Distributed Computing*, pages 148–162. Springer, 2010.

[6] Yuval Emek and Roger Wattenhofer. Stone age distributed computing. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 137–146, 2013.

[7] Ofer Feinerman and Amos Korman. Theoretical Distributed Computing Meets Biology: A Review. In Chittaranjan Hota and Pradip K. Srimani, editors, *Distributed Computing and Internet Technology*, pages 1–18, Berlin, Heidelberg, 2013. Springer.

[8] Leslie Lamport, Robert Shostak, and Marshall Pease. *The Byzantine generals problem*, page 203–226. Association for Computing Machinery, New York, NY, USA, 2019.

[9] Jennifer Lundelius and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 75–88, 1984.

[10] Danny Dolev, Joe Halpern, and H Raymond Strong. On the possibility and impossibility of achieving clock synchronization. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 504–511, 1984.

[11] Leslie Lamport and P Michael Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM (JACM)*, 32(1):52–78, 1985.

[12] Matthias Függer and Ulrich Schmid. Reconciling fault-tolerant distributed computing and systems-on-chip. *Distributed Computing*, 24:323–355, 2012.

[13] Johannes Bund, Matthias Függer, and Moti Medina. Pals: Distributed gradient clocking on chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(11):1740–1753, 2023.

[14] Brian L Partridge. The structure and function of fish schools. *Scientific american*, 246(6):114–123, 1982.

[15] Thierry Mora, Aleksandra M Walczak, Lorenzo Del Castello, Francesco Ginelli, Stefania Melillo, Leonardo Parisi, Massimiliano Viale, Andrea Cavagna, and Irene Giardina. Local equilibrium in bird flocks. *Nature physics*, 12(12):1153–1157, 2016.

[16] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Approximate consensus in highly dynamic networks: The role of averaging algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 528–539. Springer, 2015.

[17] Pantelis Pipergias Analytis, Daniel Barkoczi, Philipp Lorenz-Spreen, and Stefan Herzog. The structure of social influence in recommender networks. In *Proceedings of The Web Conference 2020*, pages 2655–2661, 2020.

[18] Bernard Chazelle. The convergence of bird flocking. *Journal of the ACM (JACM)*, 61(4):1–35, 2014.

[19] Xiaomei Liu, Shuzhi Sam Ge, and Cher-Hiang Goh. Formation potential field for trajectory tracking control of multi-agents in constrained space. *International Journal of Control*, 90(10):2137–2151, 2017.

[20] Zhenhua Pan, Chengxi Zhang, Yuanqing Xia, Hao Xiong, and Xiaodong Shao. An improved artificial potential field method for path planning and formation control of the multi-uav systems. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(3):1129–1133, 2021.

[21] Reza Olfati-Saber and Richard M Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on automatic control*, 49(9):1520–1533, 2004.

[22] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Fast, robust, quantizable approximate consensus. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

[23] Matthias Függer, Thomas Nowak, and Manfred Schwarz. Tight bounds for asymptotic and approximate consensus. *Journal of the ACM (JACM)*, 68(6):1–35, 2021.

[24] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Approximate Consensus in Highly Dynamic Networks: The Role of Averaging Algorithms, November 2014. arXiv:1408.0620 [cs] version: 2.

[25] Matthias Függer and Thomas Nowak. Fast multidimensional asymptotic and approximate consensus. In *International Symposium on DIStributed Computing (DISC) 2018*, 2018.

[26] Qun Li and Daniela Rus. Global clock synchronization in sensor networks. *IEEE Transactions on computers*, 55(2):214–226, 2006.

[27] Matthias Függer, Thomas Nowak, and Bernadette Charron-Bost. Diffusive clock synchronization in highly dynamic networks. In *2015 49th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6. IEEE, 2015.

[28] Arthur T Winfree. *The geometry of biological time*, volume 2. Springer, 1980.

[29] John Buck and Elisabeth Buck. Synchronous fireflies. *Scientific American*, 234(5):74–85, 1976.

[30] John Buck, Elisabeth Buck, James F Case, and Frank E Hanson. Control of flashing in fireflies: V. pacemaker synchronization in pteroptyx cribellata. *Journal of comparative physiology*, 144:287–298, 1981.

[31] John Jalife. Mutual entrainment and electrical coupling as mechanisms for synchronous firing of rabbit sino-atrial pace-maker cells. *The Journal of physiology*, 356(1):221–243, 1984.

[32] W Otto Friesen. Physiological anatomy and burst pattern in the cardiac ganglion of the spiny lobster panulirus interruptus. *Journal of comparative physiology*, 101(3):173–189, 1975.

[33] Ehud Sivan, Hanna Parnas, and Danny Dolev. Fault tolerance in the cardiac ganglion of the lobster. *Biological Cybernetics*, 81(1):11–23, July 1999.

[34] Renato E Mirollo and Steven H Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[35] Ariel Daliot, Danny Dolev, and Hanna Parnas. Self-Stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks. In Gerhard Goos, Juris Hartmanis, Jan Van Leeuwen, Shing-Tsaan Huang, and Ted Herman, editors, *Self-Stabilizing Systems*, volume 2704, pages 32–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. Series Title: Lecture Notes in Computer Science.

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

[37] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Pearson/Addison-Wesley, 2006. Google-Books-ID: 25p3mHu3ij8C.

[38] Nancy A. Lynch. *Distributed Algorithms*. Elsevier, April 1996.

[39] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 270–277. SIAM, 2016.

[40] Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed mis. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 217–226, 2016.

[41] Yehuda Afek, Noga Alon, Omer Barad, Eran Hornstein, Naama Barkai, and Ziv Bar-Joseph. A Biological Solution to a Fundamental Distributed Computing Problem. *Science*, 331(6014):183–185, January 2011. Publisher: American Association for the Advancement of Science.

[42] Atsushi Tero, Seiji Takagi, Tetsu Saigusa, Kentaro Ito, Dan P Bebber, Mark D Fricker, Kenji Yumiki, Ryo Kobayashi, and Toshiyuki Nakagaki. Rules for biologically inspired adaptive network design. *Science*, 327(5964):439–442, 2010.

[43] Toshiyuki Nakagaki, Makoto Iima, Tetsuo Ueda, Yasumasa Nishiura, Tetsu Saigusa, Atsushi Tero, Ryo Kobayashi, and Kenneth Showalter. Minimum-risk path finding by an adaptive amoebal network. *Physical review letters*, 99(6):068104, 2007.

[44] Vincenzo Bonifaci, Kurt Mehlhorn, and Girish Varma. Physarum can compute shortest paths. *Journal of Theoretical Biology*, 309:121–133, September 2012.

[45] Andreas Karrenbauer, Pavel Kolev, and Kurt Mehlhorn. Convergence of the non-uniform Physarum dynamics. *Theoretical Computer Science*, 816:260–269, May 2020.

[46] Shuai Xu, Wen Jiang, Xinyang Deng, and Yehang Shou. A modified *Physarum*-inspired model for the user equilibrium traffic assignment problem. *Applied Mathematical Modelling*, 55:340–353, March 2018.

[47] Wenzhong Guo and Xing Huang. Pora: A physarum-inspired obstacle-avoiding routing algorithm for integrated circuit design. *Applied Mathematical Modelling*, 78:268–286, 2020.

[48] Vincenzo Bonifaci, Enrico Facca, Frederic Folz, Andreas Karrenbauer, Pavel Kolev, Kurt Mehlhorn, Giovanna Morigi, Golnoosh Shahkarami, and Quentin Vermande. Physarum-inspired multi-commodity flow dynamics. *Theoretical Computer Science*, 920:1–20, June 2022.

[49] Anders Johannson and James Zou. A slime mold solver for linear programming problems. In *Conference on Computability in Europe*, pages 344–354. Springer, 2012.

[50] Ruben Becker, Vincenzo Bonifaci, Andreas Karrenbauer, Pavel Kolev, and Kurt Mehlhorn. Two results on slime mold computations. *Theoretical Computer Science*, 773:79–106, June 2019.

[51] Yuan Gao, Hamidreza Kamkari, Andreas Karrenbauer, Kurt Mehlhorn, and Mohammadamin Sharifi. Physarum Inspired Dynamics to Solve Semi-Definite Programs, July 2022. arXiv:2111.02291.

[52] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

[53] Shivam Garg, Kirankumar Shiragur, Deborah M. Gordon, and Moses Charikar. Distributed algorithms from arboreal ants for the shortest path problem. *Proceedings of the National Academy of Sciences*, 120(6):e2207959120, February 2023. Publisher: Proceedings of the National Academy of Sciences.

[54] Robert Hooke and Terry A Jeeves. "direct search"solution of numerical and statistical problems. *Journal of the ACM (JACM)*, 8(2):212–229, 1961.

[55] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier, 1991.

[56] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359, 1997.

[57] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee, 1995.

[58] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[59] Yanli Liu, Yuan Gao, and Wotao Yin. An improved analysis of stochastic gradient descent with momentum. *Advances in Neural Information Processing Systems*, 33:18261–18271, 2020.

[60] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[61] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[62] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[63] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[64] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[66] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[67] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):13276, 2016.

[68] Christos H Papadimitriou, Santosh S Vempala, Daniel Mitropolsky, Michael Collins, Wolfgang Maass, and Larry F Abbott. A calculus for brain computation. In *2019 Conference on Cognitive Computational Neuroscience*, 2019.

[69] William Lotter, Gabriel Kreiman, and David Cox. A neural network trained for prediction mimics diverse features of biological neurons and perception. *Nature machine intelligence*, 2(4):210–219, 2020.

[70] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[71] Volodymyr Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[72] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[73] Volodymyr Mnih. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[74] Victor Sourjik and Ned S Wingreen. Responding to chemical gradients: bacterial chemotaxis. *Current opinion in cell biology*, 24(2):262–268, 2012.

[75] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[76] HAO Jianye, Xiaotian Hao, Hangyu Mao, Weixun Wang, Yaodong Yang, Dong Li, Yan Zheng, and Zhen Wang. Boosting multiagent reinforcement learning via permutation invariant and permutation equivariant networks. In *The eleventh international conference on learning representations*, 2022.

[77] Zhuofan Xu, Benedikt Bollig, Matthias Függer, and Thomas Nowak. Permutation equivariant deep reinforcement learning for multi-armed bandit. In *2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 975–983. IEEE, 2024.

[78] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[79] D Agin. Hodgkin-huxley equations: logarithmic relation between membrane current and frequency of repetitive activity. *Nature*, 201(4919):625–626, 1964.

[80] Richard FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal*, 1(6):445–466, 1961.

[81] An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50(10):2061–2070, 2007.

[82] A Aldo Faisal, Luc PJ Selen, and Daniel M Wolpert. Noise in the nervous system. *Nature reviews neuroscience*, 9(4):292–303, 2008.

[83] Joshua H Goldwyn and Eric Shea-Brown. The what and where of adding channel noise to the hodgkin-huxley equations. *PLoS computational biology*, 7(11):e1002247, 2011.

[84] Içvara Barbier, Hadiastri Kusumawardhani, and Yolanda Schaerli. Engineering synthetic spatial patterns in microbial populations and communities. *Current Opinion in Microbiology*, 67:102149, 2022.

[85] Alex JH Fedorec, Neythen J Treloar, Ke Yan Wen, Linda Dekker, Qing Hsuan Ong, Gabija Jurkeviciute, Enbo Lyu, Jack W Rutter, Kathleen JY Zhang, Luca Rosa, et al. Emergent digital bio-computation through spatial diffusion and engineered bacteria. *Nature Communications*, 15(1):4896, 2024.

[86] M Omar Din, Tal Danino, Arthur Prindle, Matt Skalak, Jangir Selimkhanov, Kaitlin Allen, Ellixis Julio, Eta Atolia, Lev S Tsimring, Sangeeta N Bhatia, et al. Synchronized cycles of bacterial lysis for in vivo delivery. *Nature*, 536(7614):81–85, 2016.

[87] Timothy S. Gardner, Charles R. Cantor, and James J. Collins. Construction of a genetic toggle switch in Escherichia coli. *Nature*, 403(6767):339–342, January 2000. Publisher: Nature Publishing Group.

[88] Michael B. Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, January 2000. Publisher: Nature Publishing Group.

[89] Alec A. K. Nielsen, Bryan S. Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A. Strychalski, David Ross, Douglas Densmore, and Christopher A. Voigt. Genetic circuit design automation. *Science*, 352(6281):aac7341, April 2016. Publisher: American Association for the Advancement of Science.

[90] Sergi Regot, Javier Macia, Núria Conde, Kentaro Furukawa, Jimmy Kjellén, Tom Peeters, Stefan Hohmann, Eulàlia de Nadal, Francesc Posas, and Ricard Solé. Distributed biological computation with multicellular engineered networks. *Nature*, 469(7329):207–211, January 2011. Publisher: Nature Publishing Group.

[91] Alvin Tamsir, Jeffrey J. Tabor, and Christopher A. Voigt. Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'. *Nature*, 469(7329):212–215, January 2011. Publisher: Nature Publishing Group.

[92] Behzad D Karkaria, Neythen J Treloar, Chris P Barnes, and Alex JH Fedorec. From microbial communities to distributed computing systems. *Frontiers in Bioengineering and Biotechnology*, 8:834, 2020.

[93] Miha Moškon, Roman Komac, Nikolaj Zimic, and Miha Mraz. Distributed biological computation: from oscillators, logic gates and switches to a multicellular processor and neural computing applications. *Neural Computing and Applications*, 33(15):8923–8938, 2021.

[94] M Ali Al-Radhawi, Anh Phong Tran, Elizabeth A Ernst, Tianchi Chen, Christopher A Voigt, and Eduardo D Sontag. Distributed implementation of boolean functions by transcriptional synthetic circuits. *ACS Synthetic Biology*, 9(8):2172–2187, 2020.

[95] Jai P Padmakumar, Jessica J Sun, William Cho, Yangruirui Zhou, Christopher Krenz, Woo Zhong Han, Douglas Densmore, Eduardo D Sontag, and Christopher A Voigt. Partitioning of a 2-bit hash function across 66 communicating cells. *Nature Chemical Biology*, 21(2):268–279, 2025.

[96] Monica E Ortiz and Drew Endy. Engineered cell-cell communication via dna messaging. *Journal of biological engineering*, 6(1):16, 2012.

[97] Abhinav Pujar, Amit Pathania, Corbin Hopper, Amir Pandi, Cristian Ruiz Calderón, Matthias Függer, Thomas Nowak, and Manish Kushwaha. Phage-mediated intercellular crispri for biocomputation in bacterial consortia. *Nucleic Acids Research*, 53(3):gkae1256, 2025.

[98] Hadiastri Kusumawardhani, Florian Zoppi, Roberto Avendaño, and Yolanda Schaerli. Engineering intercellular communication using m13 phagemid and crispr-based gene regulation for multicellular computing in escherichia coli. *Nature communications*, 16(1):3569, 2025.

[99] John P Marken and Richard M Murray. Addressable and adaptable intercellular communication via dna messaging. *Nature Communications*, 14(1):2358, 2023.

[100] Nadrian C Seeman and Hanadi F Sleiman. Dna nanotechnology. *Nature Reviews Materials*, 3(1):1–23, 2017.

[101] Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable dna self-assembly. *Nature*, 567(7748):366–372, 2019.

[102] Alex Joesaar, Shuo Yang, Bas Bögels, Ardjan van der Linden, Pascal Pieters, B. V. V. S. Pavan Kumar, Neil Dalchau, Andrew Phillips, Stephen Mann, and Tom F. A. de Greef. DNA-based communication in populations of synthetic protocells. *Nature Nanotechnology*, 14(4):369–378, April 2019. Publisher: Nature Publishing Group.

[103] Katherine E Bujold, Aurélie Lacroix, and Hanadi F Sleiman. Dna nanostructures at the interface with biology. *Chem*, 4(3):495–521, 2018.

[104] Hui Lv, Nuli Xie, Mingqiang Li, Mingkai Dong, Chenyun Sun, Qian Zhang, Lei Zhao, Jiang Li, Xiaolei Zuo, Haibo Chen, et al. Dna-based programmable gate arrays for general-purpose dna computing. *Nature*, 622(7982):292–300, 2023.

[105] Behzad D Karkaria, Alex JH Fedorec, and Chris P Barnes. Automated design of synthetic microbial communities. *Nature communications*, 12(1):672, 2021.

[106] Miha Moškon and Miha Mraz. Programmable evolution of computing circuits in cellular populations. *Neural Computing and Applications*, 34(21):19239–19251, 2022.

[107] Da-Jung Cho, Matthias Függer, Corbin Hopper, Manish Kushwaha, Thomas Nowak, and Quentin Soubeyran. Distributed computation with continual population growth. *Distributed Computing*, pages 1–23, 2022.

[108] Matthias Függer, Thomas Nowak, and Joel Rybicki. Majority consensus thresholds in competitive lotka-volterra populations. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, pages 76–86, 2024.

[109] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 565–574, November 2000. ISSN: 0272-5428.

[110] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, PODC '04, pages 290–299, New York, NY, USA, July 2004. Association for Computing Machinery.

[111] Yuval Emek and Roger Wattenhofer. Stone age distributed computing. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, PODC '13, pages 137–146, New York, NY, USA, July 2013. Association for Computing Machinery.

[112] Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. Beeping a maximal independent set. *Distributed Computing*, 26(4):195–208, August 2013.

[113] Sabrina Rashid, Gadi Taubenfeld, and Ziv Bar-Joseph. The Epigenetic Consensus Problem. In Tomasz Jurdziński and Stefan Schmid, editors, *Structural Information and Communication Complexity*, pages 146–163, Cham, 2021. Springer International Publishing.

[114] Rud Wegscheider. Über simultane gleichgewichte und die beziehungen zwischen thermodynamik und reactionskinetik homogener systeme. *Monatshefte für Chemie und verwandte Teile anderer Wissenschaften*, 32:849–906, 1911.

[115] Martin Feinberg. *Foundations of Chemical Reaction Network Theory*, volume 202 of *Applied Mathematical Sciences*. Springer International Publishing, Cham, 2019.

[116] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, December 2008.

[117] David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, March 2010. Publisher: Proceedings of the National Academy of Sciences.

[118] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Comparing Chemical Reaction Networks: A Categorical and Algorithmic Perspective. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 485–494, New York, NY, USA, July 2016. Association for Computing Machinery.

[119] John Ross. Determination of complex reaction mechanisms. analysis of chemical, biological and genetic networks. *The Journal of Physical Chemistry A*, 112(11):2134–2143, 2008.

[120] Stephanie K Aoki, Gabriele Lillacci, Ankit Gupta, Armin Baumschlager, David Schweingruber, and Mustafa Khammash. A universal biomolecular integral feedback controller for robust perfect adaptation. *Nature*, 570(7762):533–537, 2019.

[121] Stefan Hoops, Sven Sahle, Ralph Gauges, Christine Lee, Jürgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. Copasi—a complex pathway simulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[122] Pierre Boutillier, Mutaamba Maasha, Xing Li, Héctor F Medina-Abarca, Jean Krivine, Jérôme Feret, Ioana Cristescu, Angus G Forbes, and Walter Fontana. The kappa platform for rule-based modeling. *Bioinformatics*, 34(13):i583–i592, 2018.

[123] Kiri Choi, J Kyle Medley, Matthias König, Kaylene Stocking, Lucian Smith, Stanley Gu, and Herbert M Sauro. Tellurium: an extensible python-based modeling environment for systems and synthetic biology. *Biosystems*, 171:74–79, 2018.

[124] Frank T Bergmann. Basico: a simplified python interface to copasi. *Journal of Open Source Software*, 8(90):5553, 2023.

[125] Fabricio Cravo, Matthias Függer, Thomas Nowak, and Gayathri Prakash. Mobspy: a meta-species language for chemical reaction networks. In *International Conference on Computational Methods in Systems Biology*, pages 277–285. Springer, 2022.

[126] Fabricio Cravo, Matthias Függer, and Thomas Nowak. An allee-based distributed algorithm for microbial whole-cell sensors. *npj Systems Biology and Applications*, 10(1):43, 2024.

[127] Ofer Feinerman and Amos Korman. Individual versus collective cognition in social insects. *Journal of Experimental Biology*, 220(1):73–82, 2017.

[128] Tabea Dreyer, Amir Haluts, Amos Korman, Nir Gov, Ehud Fonio, and Ofer Feinerman. Comparing cooperative geometric puzzle solving in ants versus humans. *Proceedings of the National Academy of Sciences*, 122(1):e2414274121, 2025.

[129] Christoph Grüter and Lucy Hayes. Sociality is a key driver of foraging ranges in bees. *Current Biology*, 32(24):5390–5397, 2022.