Adaptive Collective Responses to Local Stimuli in Anonymous Dynamic Networks

Shunhao Oh
School of Computer Science, Georgia Institute of Technology

Dana Randall
School of Computer Science, Georgia Institute of Technology

Andréa W. Richa
School of Computing and Augmented Intelligence, Arizona State University

Abstract

We develop a framework for self-induced phase changes in programmable matter in which a collection of agents with limited computational and communication capabilities can collectively perform appropriate global tasks in response to local stimuli that dynamically appear and disappear. Agents are represented by vertices in a dynamic graph $\mathcal{G}$ whose edge set changes over time, and stimuli are placed adversarially on the vertices of $\mathcal{G}$ where each agent is only capable of recognizing a co-located stimulus. Agents communicate via token passing along edges to alert other agents to transition to an Aware state when stimuli are present and an Unaware state when the stimuli disappear. We present an Adaptive Stimuli Algorithm that can handle arbitrary adversarial stimulus dynamics, while an adversary (or the agents themselves) reconfigure the connections (edges) of $\mathcal{G}$ over time in a controlled way. This algorithm can be used to solve the *foraging problem* on reconfigurable graphs where, in addition to food sources (stimuli) being discovered, removed, or shifted arbitrarily, we would like the agents to consistently self-organize, using only local interactions, such that if the food remains in a position long enough, the agents transition to a *gather phase* in which many collectively form a single large component with small perimeter around the food. Alternatively, if no food source has existed recently, the agents should undergo a self-induced collective phase change and switch to a *search phase* in which they distribute themselves randomly throughout the graph to search for food. Unlike previous approaches to foraging, this process is indefinitely repeatable, withstanding competing broadcast waves of state transition that may interfere with each other. Like a physical phase change, such as the ferromagnetic models underlying the gather and search algorithms used for foraging, microscopic changes in the environment trigger these macroscopic, system-wide transitions as agents share information and respond locally to get the desired collective response.

2012 ACM Subject Classification Theory of computation-Self-organization; Theory of computation-Random walks and Markov chains

Keywords and phrases Dynamic networks, adaptive stimuli, foraging, self-organizing particle systems, programmable matter

Funding *Shunhao Oh*: Supported by the National Science Foundation (NSF) award CCF-2106687 and by U.S. Army Research Office (ARO) award MURI W911NF-19-1-0233.
Dana Randall: Supported by the National Science Foundation (NSF) award CCF-2106687 and by U.S. Army Research Office (ARO) award MURI W911NF-19-1-0233.
Andréa W. Richa: Supported in part by the National Science Foundation (NSF) award CCF-2106917 and by U.S. Army Research Office (ARO) award MURI W911NF-19-1-0233.

1 Introduction

Self-organizing collective behavior of interacting agents is a fundamental, nearly ubiquitous phenomenon across fields, reliably producing rich and complex coordination. In nature, examples at the micro- and nano-scales include coordinating cells, including our own immune system or self-repairing tissue (e.g., [1]), and bacterial colonies (e.g., [31, 37]); at the macro-scale it can represent flocks of birds [9], shoals of fish aggregating to intimidate predators [33], fire ants forming rafts to survive floods [35], and human societal dynamics such as segregation [40]. Common characteristic of these disparate systems is that they are all self-actuated and respond to simple, local environmental stimuli to collectively change the ensemble's behavior.

In 1991, Toffoli and Margolus coined *programmable matter* to realize a physical computing medium composed of simple, homogeneous entities that can dynamically alter its physical properties in a programmable fashion, controlled either by user input or its own autonomous sensing of its environment [42]. There are formidable challenges to realizing such collective tasks and many researchers in distributed computing and swarm and modular robotics have investigated how such small, simply programmed entities can coordinate to solve complex tasks and exhibit useful emergent collective behavior (e.g., [38]). A more ambitious goal, suggested by self-organizing collective systems in nature, is to design programmable matter systems of *self-actuated* individuals that *autonomously respond to continuous, local changes in their environment.*

The Dynamic Stimuli Problem: As a distributed framework for a large number of agents collectively self-organizing in response to changing stimuli, we consider the *dynamic stimuli problem.* These agents have limited computational capabilities and each only communicates with a small set of immediate neighbors. We represent these n agents via an edge-dynamic graph $\mathcal{G}$ on n vertices, where each vertex of $\mathcal{G}$ represents an agent and an edge represents a pair of agents that can perceive and interact with each other while the edge is present in $\mathcal{G}$. At arbitrary points in time, *stimuli* dynamically appear and disappear at the vertices of $\mathcal{G}$, possibly adversarially — these can represent a threat, such as an unexpected predator, or an opportunity, such as new food or energy resources.

The agents each independently transition through states that are updated according to their current knowledge of the world that dictate what actions they should perform. An agent present at the same vertex as a stimulus acts as a *witness* and alerts other agents via the edges of $\mathcal{G}$. If any agent continues to witness some stimulus over an extended period of time, we want all agents to eventually be alerted, switching to the Aware state; on the other hand, once witnesses stop sensing a stimulus for long enough, all agents should return to the Unaware state. Such collective state changes may repeat indefinitely as stimuli appear and disappear over time. Converging to these two global states enables agents to carry out *differing behaviors in the presence or absence of stimuli*, as observed by the respective witnesses. As a notable and challenging example, in *the foraging problem,* "food" may appear or disappear at arbitrary locations in space over time and we would like the collective to *gather around food* (also known as *dynamic free aggregation*) or *disperse in search of new sources*, depending on whether or not an active food source has been identified (see, e.g., Section 3.2.1 of [5] and [30]). Cannon *et al.* [7] showed how computationally limited agents can coordinate their movements to gather or disperse on the triangular lattice; however there the desired goal, aggregation or dispersion, is fixed in advance and the algorithm cannot easily be adapted to move between these according to changing needs.

In our framework, we assume that, at any point in time, stimuli is located in at most

$s_{\max}$ vertices of $\mathcal{G}$, where $s_{\max}$ is a constant. Agents are anonymous and each acts as a finite automaton with constant-size memory, constant degree, and no access to global information other than $s_{\max}$ and a constant upper bound Δ on the maximum degree. Individual agents are activated according to their own Poisson clocks and perform instantaneous actions upon activation, a standard way to allow sites to update independently (since the probability that two Poisson clocks tick at the exact same instant in time is negligible). For simplicity, we assume the Poisson clocks have the same rate: This model is equivalent to a *random sequential scheduler* that chooses an agent uniformly at random to be activated at discrete iterations $t \in \{1, 2, 3, \ldots\}$ (A brief discussion on non-uniform Poisson clock rates and other generalizations appears in Section 7).

In addition to the stimuli dynamics, we assume that the *connections (edges)* of the dynamic graph $\mathcal{G}$ are *reconfigured over time* in a controlled way that still allows the agents to successfully manage the waves of state changes. In a nutshell, we say that a dynamic graph $\mathcal{G}$ is *reconfigurable* if it maintains recurring local connectivity of the Aware agents as its edge set changes—i.e., if the 1-hop aware neighborhood of each Aware agent induces a connected subgraph at any point in time in $\mathcal{G}$ (see Definition 11). The edge dynamics may be fully in the control of an abstract adversary (with access to limited information on the agents' states), or may be controlled by the agents themselves, depending on the context (e.g., in the foraging problem presented in Section 5, the agents control the edge dynamics). While the constraints on edge dynamics may seem restrictive, we note that they are naturally satisfied by our proposed algorithm for solving the general foraging problem in Section 5; in the future, it would be interesting to find other application scenarios where reconfigurable graphs naturally arise.

We denote the *configuration* of the system at iteration t by the pair $\sigma_t = (G_t, \theta_t)$, where $G_t = (V, E_t)$ is a snapshot of the underlying (undirected) dynamic graph $\mathcal{G}$ at iteration t, with set of nodes (agents) V and set of edges E_t, and where $\theta_t(u)$ gives the state of the agent u at iteration t, for each $u \in V$. When *activated* at iteration t, an agent perceives its own state and the states of its current neighbors in G_t, performs a bounded amount of computation, and can change its own and its neighbors states, including any "tokens" they may have (i.e., constant size messages exchanged between agents).[1] At each iteration $t \in \{1, 2, 3, \ldots\}$, we denote by $\mathcal{S}_t \subseteq V$ the set of stimuli, where $|\mathcal{S}_t| \leq s_{\max}$. The sets $\mathcal{S}_t$ can change arbitrarily (i.e., by an adaptive adversary) over time. We will formally define what it means for a dynamic graph $\mathcal{G}$ given by the sequence $(G_0, G_1, \ldots)$ to be *reconfigurable* (with respect to $(\theta_0, \theta_1, \ldots)$) in Section 4.

Overview of Results: Our contributions are two-fold. First, we present an *efficient algorithm for the dynamic stimuli problem for the class of reconfigurable dynamic graphs.* Whenever an agent encounters a new stimulus, the entire collective efficiently transforms to the Aware state, so the agents can implement an appropriate collective response. After a stimulus vanishes, they all return to the Unaware state, recovering their "neutral" collective behavior.

We show that the system will always converge to the appropriate state (i.e., Aware or Unaware) once the stimuli stabilize for a sufficient period of time. Specifically, if there are no stimuli in the system for a sufficient period of time, then all agents in the Adaptive Stimuli Algorithm will have reached and remain in the Unaware state in $O(n^2)$ expected

[1] Since we assume a sequential scheduler, such an action can be justified; in the presence of a stronger adversarial scheduler, e.g., the asynchronous scheduler, one would need a more detailed message passing mechanism to ensure the transfer of tokens between agents, and resulting changes in their states.

time (Theorem 15). Likewise, if the set of stimuli remains unchanged for a sufficient period of time, all agents will have reached and remain in the Aware state in expected time that is a polynomial in n and the "recurring rate" of G, which captures how frequently disconnected vertices come back in contact with each other (Theorem 16). In particular, if G is static or if G_t is connected, for all t, the expected convergence time until all agents transition to the Aware state is $O(n^4)$ (Theorem 4) and $O(n^6 \log n)$ (Corollary 17), respectively. Moreover, the system can recover if the stimulus set changes before the system fully converges to the Aware or Unaware states.

While the arguments would have been simpler for sequences of connected graphs generated by, say, an oblivious adversary, the extension to the broader class of reconfigurable graphs includes graphs possibly given by non-oblivious adversaries that may occasionally disconnect. This generalization provides more flexibility for agents in the Aware and Unaware states to implement more complex behaviors, as was needed in the application of the Adaptive Stimuli Algorithm to foraging, which we describe next.

Our second main contribution is the first efficient algorithm for the *foraging problem*, where food dynamically appears and disappears over time at arbitrary sites on a finite $\sqrt{N} \times \sqrt{N}$ region of the triangular lattice, where $n \leq N$. Agents want to gather around any discovered food source (also known as *dynamic free aggregation*) or disperse in search of food. The algorithm of Cannon *et al.* [7, 30] uses insights from the high and low temperature phases of the ferromagnetic Ising model from statistical physics [4] to provably achieve either desired collective response: There is a preset global parameter λ related to inverse temperature, and the algorithm provably achieves aggregation when λ is sufficiently high and dispersion when λ is sufficiently low. We show here that by applying the Adaptive Stimuli Algorithm, the phase change (or bifurcation) for aggregation and dispersion can be self-modulated based on local environmental cues that are communicated through the collective to induce desirable system-wide behaviors in polynomial time in n and N, as stated in Theorems 24 and 25.

In the context of foraging, collectively transitioning between Aware and Unaware states enables agents to *correctly self-regulate system-wide adjustments* in their bias parameters when one or more agents notice the presence or depletion of food to induce the appropriate global coordination to provably transition the collective between macro-modes when required. We believe other collective behaviors exhibiting emergent bifurcations, including separation/integration [6] and alignment/nonalignment [27], can be similarly self-modulated.

The distinction between polynomial and super-polynomial running times is significant here because our algorithms necessarily make use of competing broadcast waves to propagate commands to change states. A naive implementation of such a broadcast system may put us in situations where neither type of wave gets to complete its propagation cycle. This may continue for an unknown amount of time, so the agents may fail to reach an agreement on their state. Our carefully engineered token passing mechanism ensures that when a stimulus has been removed, the *rate at which the agents "reset" to the Unaware state outpaces the rate at which the cluster of Aware agents may continue to grow*, ensuring that the newer broadcast wave always supersedes previous ones and completes in expected polynomial time. Moreover, while a stimulus is present, there is a continuous *probabilistic generation of tokens that move according to a d_{max}-random walk* among Aware agents until they find a new agent to become Aware, thus ensuring the successful convergence to the Aware state in expected polynomial time.

Of independent interest is the extension of the arguments in [7] to the setting where there exists a single immobile agent, namely the one that finds the food source. The compression algorithm of [7] specifies a restricted set of local movements on the triangular lattice that guarantees that the configuration remains simply connected. While preserving connectivity is a straightforward consequence of the choice of moves, the proof that all simply connected configurations are reachable by these moves (i.e., the irreducibility of the Markov chain) becomes significantly more challenging. This proof, given in Section 6, builds on ideas from [7] and is similarly long and technical.

Related work: Dynamic networks have been of growing interest recently and have spawned several model variants (see, e.g., the surveys in [8] and [29]). There is also a vast literature on broadcasting, or information dissemination, in both static and dynamic networks (e.g., [28, 25, 10, 20, 19]), where one would like to disseminate k messages to all nodes of a graph G, usually with unique token ids and $k \leq n$, polylog memory at the nodes (which may also have unique ids), and possibly nodes' knowledge of k and/or n. Note that any of these assumptions violates our agents' memory or computational capabilities. Moreover, since our collective state-changing process runs indefinitely, any naive adaptations of these algorithms would need that $k \to \infty$ to ensure that with any sequence of broadcast waves, the latest always wins.

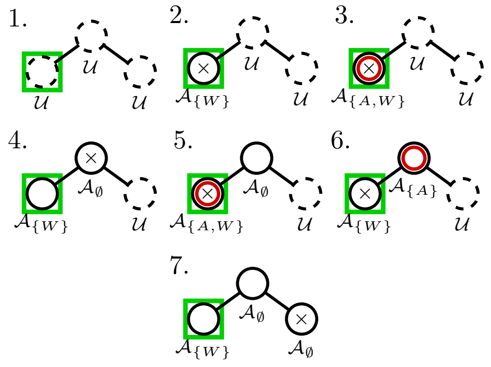
Broadcast algorithms that do not explicitly keep any information on k (i.e., the number of tokens, or of corresponding broadcast waves) or n would be more amenable to our agents. Amnesiac flooding is one such broadcast algorithm that works on a network of anonymous nodes without keeping any state or other information as the broadcast progresses. In [26], Hussak and Trehan show that amnesiac flooding will always terminate in a static network under synchronous message passing, but may fail on a dynamic network or in non-synchronous executions.

Many studies in self-actuated systems take inspiration from emergent behavior in social insects, but either lack rigorous mathematical foundations explaining the generality and limitations as sizes scale (see, e.g., [24, 23, 11, 44]), often approaching the thermodynamic limits of computing [43] and power [12], or rely on long-range signaling, such as microphones or line-of-sight sensors [41, 21, 22, 36]. Some recent work on stochastic approaches modeled after systems from particle physics has been made rigorous, but only when a single, static goal is desired [7, 6, 30, 2, 39, 27].

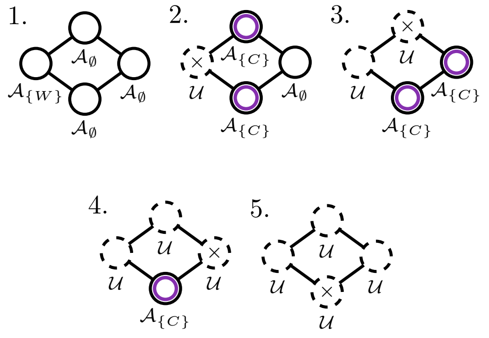
2 The Adaptive Stimuli Algorithm

The *Adaptive Stimuli Algorithm* is designed to efficiently respond to dynamic local stimuli that indefinitely appear and disappear at the vertices of $\mathcal{G}$. Recall the goal of this algorithm is to allow the collective to converge to the AWARE state whenever a stimulus is witnessed for long enough and to the UNAWARE state if no stimulus has been detected recently. The algorithm converges in expected polynomial time in both scenarios under a reconfigurable dynamic setting, as we show in Sections 3-4, even as the process repeats indefinitely.

All agents know two constant parameters of the system: $\Delta \geq 1$, an upper bound on the maximum degree of the graph, and $s_{\max} \geq 1$, an upper bound on the size of the stimulus sets $\mathcal{S}_t$ at all times t (which is needed to determine the constant probability $p < 1/s_{\max}$ that agents use to change states or generate certain tokens). Our algorithm defines a carefully balanced token passing mechanism, where a *token* is a constant-size piece of information: Upon activation, an AWARE witness u continuously generates *alert tokens* one at a time, with probability p, which will each move through a *random walk over* AWARE *agents until they come in contact with a neighboring* UNAWARE *agent* v: The token is then consumed and v changes its state to AWARE. On the other hand, if a witness notices that its co-located stimulus has disappeared, it will initiate an *all-clear token broadcast wave* which will proceed



(a) The leftmost agent senses a stimulus and sets its witness flag. It then gradually converts all agents to the AWARE state through the distribution of alert tokens. Each alert token follows a random walk over AWARE agents.



(b) An agent with a set witness flag that no longer senses a stimulus broadcasts a wave of all-clear tokens to all agents, converting them to the UNAWARE state.

Figure 1 Step-by-step illustrations of possible sequences of agent activations in the presence and absence of stimuli respectively. The agent being activated is denoted by "×," stimuli (if present) are denoted by a green squares, AWARE ($\mathcal{A}$) and UNAWARE ($\mathcal{U}$) agents are denoted by solid and dashed circles respectively, and alert and all-clear tokens are denoted by red and purple inner circles respectively. State subscripts correspond to the flags as described in Section 2.

through agents in the AWARE state, switching those to UNAWARE. Figures 1a and 1b illustrate these two scenarios (where flags and agent labels will be defined later in this section).

The differences between these two carefully crafted token passing mechanisms ensure that whenever waves of alert tokens and all-clear tokens compete to convert agents to the AWARE and UNAWARE states respectively, the UNAWARE broadcast wave will always outpace any residual AWARE waves in the system (in terms of the expected rates at which these waves spread). In the case where there has been no stimulus for long enough, this allows the collective to converge to the desired UNAWARE state. In the case where the stimulus set is non-empty and remains stable for long enough, this allows the collective to shed all traces of all-clear tokens as any cluster of AWARE agents containing them will quickly dissipate (and UNAWARE agents never generate tokens), so that eventually only AWARE waves remain and allow the collective to converge to the desired AWARE state.

We utilize the following agent flags in the Adaptive Stimuli Algorithm:

- **Alert token flag** (A): this flag is set if the agent has an alert token.
- **All-clear token flag** (C): this flag is set if the agent has an all-clear token.
- **Witness flag** (W): this flag is set if the agent witnesses a stimulus.

There are *two states* an agent can be in: The UNAWARE state $\mathcal{U}$, and the AWARE state $\mathcal{A}$. We further refine the AWARE state by adding subscripts that denote which flags are currently set at the agent, i.e., an AWARE agent will be in exactly one of $\mathcal{A}_\emptyset$, $\mathcal{A}_{\{A\}}$, $\mathcal{A}_{\{W\}}$, $\mathcal{A}_{\{A,W\}}$, or $\mathcal{A}_{\{C\}}$ states. Note that the all-clear token flag is only set when the other two are not, giving us these six distinct states in total.

Algorithm 1 formalizes the actions executed by each agent u when activated, which depend on u's current state, including its set $N_{\mathcal{A}}(u)$ of aware neighbors, and whether it senses a stimulus. Figures 1a and 1b illustrate the execution of our algorithm at a high level, upon a stimulus appearing or disappearing in G, respectively, showing how tokens "move" through the graph and affect the agents' states. We describe the behavior of an activated agent u for each of the possible cases below:

- **Non-matching witness flag**: This is a special case that occurs if u is currently a witness to some stimuli but its witness flag has not been set yet, or if u has its witness flag set but no longer witnesses stimuli. This case takes priority over all the other possible cases, since u cannot take any action before its witness flag is up to date with its situation. If u witnesses a stimulus but does not have the witness flag set, then with probability p, switch u to state $\mathcal{A}_{\{W\}}$. On the other hand, if u does not witness stimuli but has the witness flag set, switch u to UNAWARE (by setting $u.state = \mathcal{U}$) and broadcast an all-clear token to all of u's AWARE neighbors (this token overrides any alert tokens the neighbors may have).
- **Unaware state ($\mathcal{U}$)**: If u has an AWARE neighbor v with an alert token (i.e., $v.state \in \{\mathcal{A}_{\{A\}}, \mathcal{A}_{\{A,W\}}\}$), then u consumes the alert token from v (by setting v's alert token flag to FALSE) and switches to the state $\mathcal{A}_{\emptyset}$.
- **Aware state with alert token ($\mathcal{A}_{\{A\}}$, $\mathcal{A}_{\{A,W\}}$)**: Pick a random neighbor of u such that each neighbor is picked with probability $\frac{1}{\Delta}$, with a probability $1 - \frac{d_G(u)}{\Delta}$ of staying at u (this executes a d_{max}-random walk [16]). If an AWARE neighbor v is picked and v does not hold an alert nor an all-clear token (that is, $v.state \in \{\mathcal{A}_{\emptyset}, \mathcal{A}_{\{W\}}\}$), move the alert token to v by toggling the alert token flags on both u and v.
- **Aware state with witness flag only: ($\mathcal{A}_{\{W\}}$)**: With probability p, u generates a new alert token and sets its corresponding flag, switching to state $\mathcal{A}_{\{A,W\}}$.
- **Aware state with all-clear token ($\mathcal{A}_{\{C\}}$)**: Switch u to the UNAWARE state $\mathcal{U}$ and broadcast the all-clear token to all of its AWARE state neighbors.

The use of a d_{max}-random walk instead of regular random walk (Line 18 of Algorithm 1) normalizes the probabilities of transitioning along an edge by the maximum degree of the nodes (or a constant upper bound on that), so that these transition probabilities cannot change during the evolution of the graph. A d_{max}-random walk has polynomial hitting time on any connected dynamic network (while a regular random walk might not) [3].

3 Static graph topologies

In this section, we prove the correctness of Algorithm 1 for *static connected graph topologies* (Theorems 3 and 4), where the edge set never changes and the dynamics are only due to the placement of stimuli. In Section 4, we show that the same proofs apply with little modification to the *reconfigurable case* as well.

We will first define the *state invariant*, which will always hold given that we start from an initial configuration that satisfies the invariant, as we show in Lemma 2. Since the invariant is trivially satisfied for our initial configuration with all agents are in the UNAWARE state, the state invariant will hold throughout the execution of Algorithm 1. In the remainder of this paper, a *component* will refer to a connected component of the subgraph induced in G by the set of AWARE agents. The state invariant requires every component (of AWARE agents) to contain an agent that either holds a all-clear token or has its witness flag set. This guarantees that each AWARE agent either remains connected to a witness agent in G or will eventually be converted to the UNAWARE state.

▶ **Definition 1** (State Invariant). *We say a component satisfies the* state invariant *if it contains at least one agent in the states $\mathcal{A}_{\{W\}}$, $\mathcal{A}_{\{A,W\}}$ or $\mathcal{A}_{\{C\}}$. A configuration satisfies the state invariant if every component (if any) of the configuration satisfies the state invariant.*

Algorithm 1 Adaptive Stimuli Algorithm.

```
procedure THE ADAPTIVE-STIMULI-ALGORITHM($u$)
    Let $p < (1/s_{\max})$, $p \in (0,1)$
    $u.isWitness \leftarrow$ TRUE if $u$ witnesses a stimulus, else $u.isWitness \leftarrow$ FALSE
    if $u.isWitness$ and $u.state \notin \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}\}$ then
        With probability $p$, $u.state \leftarrow \mathcal{A}_{\{W\}}$    ▷ $u$ becomes AWARE witness with prob. $p$
    else if $\neg u.isWitness$ and $u.state \in \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}\}$ then ▷ stimulus no longer at $u$
        for each $v \in N_{\mathcal{A}}(u)$ do    ▷ $N_{\mathcal{A}}(u)$ = current AWARE neighbors of $u$
            $v.state \leftarrow \mathcal{A}_{\{C\}}$    ▷ all-clear token broadcast to AWARE neighbors of $u$
        $u.state \leftarrow \mathcal{U}$
    else
        switch $u.state$ do
            case $\mathcal{U}$:
                if $\exists v \in N_{\mathcal{A}}(u), v.state = \mathcal{A}_S \in \{\mathcal{A}_{\{A\}}, \mathcal{A}_{\{A,W\}}\}$ then    ▷ $v$ has alert token
                    $v.state \leftarrow \mathcal{A}_{S \setminus \{A\}}$    ▷ $v$ consumes alert token
                    $u.state \leftarrow \mathcal{A}_{\emptyset}$    ▷ $u$ becomes aware
            case $\mathcal{A}_{\{A\}}$ or $\mathcal{A}_{\{A,W\}}$:
                $x \leftarrow$ random value in $[0,1]$
                if $x \leq d_G(u)/\Delta$ then    ▷ $d_{max}$-random walk
                    $v \leftarrow$ random neighbor of $u$
                    if $v.state = \mathcal{A}_{S'} \in \{\mathcal{A}_{\emptyset}, \mathcal{A}_{\{W\}}\}$ then    ▷ AWARE state, no alert token
                        Let $u.state = \mathcal{A}_S$; $u.state \leftarrow \mathcal{A}_{S \setminus \{A\}}$    ▷ $u$ sends alert token to $v$
                        $v.state \leftarrow \mathcal{A}_{S' \cup \{A\}}$    ▷ $v$ receives alert token
            case $\mathcal{A}_{\{W\}}$:
                With probability $p$, $u.state \leftarrow \mathcal{A}_{\{A,W\}}$ ▷ generate alert token with prob. $p$
            case $\mathcal{A}_{\{C\}}$:
                for each $v \in N_{\mathcal{A}}(u)$ do
                    $v.state \leftarrow \mathcal{A}_{\{C\}}$    ▷ $u$ broadcasts all-clear token to all aware neighbors
                $u.state \leftarrow \mathcal{U}$
```

▶ **Lemma 2.** *If the current configuration satisfies the state invariant, then all subsequent configurations reachable by Algorithm 1 also satisfy the state invariant.*

Proof. We will prove by induction, starting from a configuration where the state invariant currently holds. Let u be the next agent to be activated. If u witnesses a stimulus but $u.state \notin \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}\}$, switching u to state $\mathcal{A}_{\{W\}}$ does not affect the state invariant. (Note that u can only enter state $\mathcal{A}_{\{A,W\}}$ from $\mathcal{A}_{\{W\}}$.) Conversely, if u does not witness stimuli, but $u.state \in \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}\}$ switching u to state $\mathcal{U}$ can potentially split the component it is in into multiple components. However, as all neighbors of u will also be set to state $\mathcal{A}_{\{C\}}$, each of these new components will contain an agent in state $\mathcal{A}_{\{C\}}$.

Otherwise, if $u.state = \mathcal{U}$, it only switches to the AWARE state if it neighbors another AWARE agent. As the component u joins must contain an agent in states $\mathcal{A}_{\{W\}}$, $\mathcal{A}_{\{A,W\}}$ or $\mathcal{A}_{\{C\}}$, the new configuration will continue to satisfy the state invariant. If $u.state = \mathcal{A}_{\{C\}}$, similar to the case where u does not witness stimuli but $u.state \in \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}\}$, activating u may split the component it is in. As before, all neighbors of u will be set to state $\mathcal{A}_{\{C\}}$, so each of these newly created components satisfy the state invariant. The remaining possible cases only toggle the alert token flag, which does not affect the state invariant. ◀

This allows us to state our main results in the static graph setting. We let $T \in \mathbb{N}$ represent the time when the stimuli becomes stable for long enough to converge (where T is unknown to the agents). Without loss of generality, for the sake of our proofs, we will assume that the stimuli set $\mathcal{S}_t = \mathcal{S}_T$, for all $t \geq T$, although we just need the stimuli set to stay stable for long enough (i.e., for $O(n^2)$ and $O(n^4)$ expected time, respectively) for the two theorems below that show efficient convergence.

▶ **Theorem 3.** *Starting from any configuration satisfying the state invariant over a static connected graph topology G, if $|\mathcal{S}_T| = 0$, then all agents will reach and remain in the* Unaware *state in $O(n^2)$ expected iterations, after time T.*

▶ **Theorem 4.** *Starting from any configuration satisfying the state invariant over a static connected graph topology G, if $|\mathcal{S}_T| > 0$, then all agents will reach and stay in the* Aware *state in $O(n^4)$ expected iterations, after time T.*

For the rest of this section, we will be establishing the definitions and lemmas used to prove Theorems 3 and 4. The proofs of these theorems rely on carefully eliminating residual Aware agents from previous broadcasts. These agents, which we call residuals (Definition 5), have yet to receive an all-clear token and thus will take some time before they can return to the Unaware state.

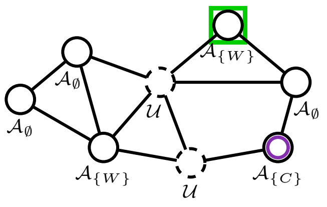


Figure 2 A configuration with two residual components. Green squares denote stimuli. The component on the left is residual because it contains an agent in state $\mathcal{A}_{\{W\}}$ that does not observe a stimulus; the component on the right is residual as it contains an agent in state $\mathcal{A}_{\{C\}}$.

▶ **Definition 5** (Residuals). *A* residual component *is a component that satisfies at least one of the following two criteria:*
1. *It contains an agent in state $\mathcal{A}_{\{C\}}$.*
2. *It contains an agent in state $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$ that does not witness a stimulus.*

We call agents belonging to residual components residuals*.*

When starting from an arbitrary configuration satisfying the state invariant, it is likely that there will be a large number of residual components. As these components may not be connected to any stimulus, so we would want to clear them out by returning their agents to the Unaware state. Keeping residual components around can also be problematic as the presence of residuals may cause even more residuals to form. Furthermore, later on when we allow the graph to reconfigure itself, residuals may obstruct Unaware state agents from coming into contact with non-residual components (which we want to grow), which may prevent alert tokens from reaching them. Our main tool to show that all residuals eventually vanish is a *potential function* that decreases faster than it increases.

▶ **Definition 6** (Potential). *For a configuration σ, we define its potential $\Phi(\sigma)$ as*

$$\Phi(\sigma) := \Phi_{\mathcal{A}}(\sigma) + \Phi_{\mathcal{A}_{\{A\}}}(\sigma)$$

where $\Phi_{\mathcal{A}}(\sigma)$ and $\Phi_{\mathcal{A}_{\{A\}}}(\sigma)$ represent total the number of Aware *agents and the number of* Aware *agents with an alert token respectively.*

We use the following lemma to guarantee, in Lemma 8, that the expected number of steps before all residuals are removed is polynomial.

▶ **Lemma 7.** *Assume $n \geq 2$, and let $0 < \eta < 1$. Consider two random sequences of probabilities $(p_t)_{t\in\mathbb{N}_{\geq 0}}$ and $(q_t)_{t\in\mathbb{N}_{\geq 0}}$, with the properties that $\frac{1}{n} \leq p_t \leq 1$ and $0 \leq q_t \leq \frac{\eta}{n}$, and $p_t + q_t \leq 1$. Now consider a sequence $(X_t)_{t\in\mathbb{N}_{\geq 0}}$, where*

$$X_{t+1} \begin{cases} \leq X_t - 1 & \textit{with probability } p_t \\ = X_t + 1 & \textit{with probability } q_t \\ = X_t & \textit{with probability } 1 - p_t - q_t. \end{cases}$$

Then $\mathbb{E}\left[\min\{t \geq 0 \mid X_t = 0\}\right] \leq \frac{nX_0}{1-\eta}$.

Proof. For each $k \in \mathbb{N}_{\geq 0}$, we define a random sequence $(Y_t^{(k)})_{t\in\mathbb{N}_{\geq 0}}$ such that $Y_0^{(k)} = k$ and

$$Y_{t+1}^{(k)} \begin{cases} = Y_t^{(k)} - 1 & \text{with probability } \frac{1}{n} \\ = Y_t^{(k)} + 1 & \text{with probability } \frac{\eta}{n} \\ = Y_t^{(k)} & \text{otherwise.} \end{cases}$$

For each such k, let $S_k := \mathbb{E}\left[\min\{t \geq 0 \mid Y_t^{(k)} = 0\}\right]$. Let $T_{k+1}^{(k)} = 0$ and for $i \in \{k, k-1, \ldots, 1\}$, let $T_i^{(k)} = \min\{t \geq 0 \mid Y_t^{(k)} \leq i\} - T_{i+1}^{(k)}$ denote the number of time steps after $T_{i+1}^{(k)}$ before the first time step t where $Y_t^{(k)} \leq i$. We observe that each $T_i^{(k)}$ is identically distributed, with $\mathbb{E}T_i^{(k)} = \mathbb{E}T_1^{(k)} = S_1$. Also, we observe that $S_k = \mathbb{E}[\sum_{i=1}^k T_i^{(k)}]$, so $S_k = k \cdot S_1$ for all k. We can thus compute S_1 by conditioning on the first step:

$$S_1 = \frac{1}{n}(1) + \frac{\eta}{n}(S_2 + 1) + \left(1 - \frac{1+\eta}{n}\right)(S_1 + 1) = 1 + \frac{\eta}{n} \cdot 2S_1 + \left(1 - \frac{1+\eta}{n}\right) S_1$$

This implies $S_1 = \frac{n}{1-\eta}$ and thus $\mathbb{E}\left[\min\{t \geq 0 \mid Y_t^{(k)} = 0\}\right] = S_k = \frac{kn}{1-\eta}$.

We can then couple $(X_t)_{t\in\mathbb{N}_{\geq 0}}$ and $(Y_t^{(X_0)})_{t\in\mathbb{N}_{\geq 0}}$ in a way such that $Y_{t+1}^{(X_0)} = Y_t^{(X_0)} + 1$ whenever $X_{t+1} > X_t$, and $Y_{t+1}^{(X_0)} = Y_t^{(X_0)} - 1$ whenever $X_{t+1} < X_t$. We thus have $Y_t^{(X_0)} \geq X_t$ always, and so $\mathbb{E}\left[\min\{t \geq 0 \mid X_t = 0\}\right] \leq \mathbb{E}\left[\min\{t \geq 0 \mid Y_t^{(X_0)} = 0\}\right] \leq \frac{kX_0}{(1-\eta)}$. ◀

We show in Lemma 8 that after a polynomial number of steps in expectation, we will reach a configuration with no residual components, where p is as defined in Algorithm 1.

▶ **Lemma 8.** *We start from a configuration satisfying the state invariant over a static connected graph G and assume that there are no more than s_{max} stimuli at any point of time. Then the expected number of steps before we reach a configuration with no residual components is at most $2n^2/(1 - ps_{max})$.*

Proof. We apply Lemma 7 to the sequence of potentials $(\Phi(\sigma_t))_{t\in\mathbb{N}_{\geq 0}}$ where σ_t is the configuration after iteration t. As long as there exists a residual component, there will be at least one agent that will switch to the UNAWARE state upon activation. This gives a probability of at least $1/n$ in any iteration of decreasing the current potential by at least 1.[2]

[2] Note that in Algorithm 1, u can switch from state $\mathcal{A}_{\{A,W\}}$ to $\mathcal{U}$, decreasing Φ by 2.

There are only two ways for the potential to increase. The first is when a new alert token is generated by a witness, and the second is when an UNAWARE witness switches to an AWARE state. The activation of a witness thus increases the current potential by exactly 1, with probability p. As there are at most $s_{\max}$ witnesses observing stimuli, this happens with probability at most $ps_{\max}/n < 1/n$. Note that the consumption of an alert token to add a new AWARE state agent to a residual component does not change the current potential. Neither does switching agents to the all-clear token state affect the potential.

By Lemma 7, as $\Phi(\sigma_0) \leq 2n$, within $2n^2/(1 - ps_{\max})$ steps in expectation, we will either reach a configuration σ with $\Phi(\sigma) = 0$, or a configuration with no residual components, whichever comes first. Note that if $\Phi(\sigma) = 0$, then σ cannot have any residual components, completing the proof. ◀

We now show that as long as no agent is removed from the stimulus set, after all residual components are eliminated, no new ones will be generated:

▶ **Lemma 9.** *We start from a configuration satisfying the state invariant over a static connected graph G, and assume that no agent will be removed from the stimulus set from the current point on. If there are currently no residuals, then a residual cannot be generated.*

Proof. With no residual components in the current iteration, there will be no agents in state $\mathcal{A}_{\{C\}}$, and all agents in state $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$ will be witnessing stimuli. This means that no agent on activation will switch another agent to state $\mathcal{A}_{\{C\}}$. All agents in states $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$ will continue to witness stimuli by assumption of the lemma, and agents will only switch to states $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$ if they are witnessing stimuli. Thus no component will be residual in the next iteration. ◀

In the case where $|\mathcal{S}_T| = 0$, as long as the state invariant holds, having no residuals and no stimuli means that all agents must be in the UNAWARE state. Combining Lemmas 8 and 9 thus gives us Theorem 3. Furthermore, if $|\mathcal{S}_T| > 0$, Lemmas 8 and 9 also show that eventually all residuals will be eliminated. No AWARE state agent can switch back to the UNAWARE state following this, as that would require either an all-clear token to be generated (creating a residual) or a stimulus to vanish. This allows us to apply Lemma 10 repeatedly (for each UNAWARE agent) to show Theorem 4.

▶ **Lemma 10.** *We start from a configuration satisfying the state invariant over a static connected graph G, and assume that there are no residual agents, the stimulus set is nonempty and no agent will be removed from the stimulus set from the current point on. Then the expected number of iterations before the next agent switches from the UNAWARE to the AWARE state is at most $O(n^3)$.*

Proof. In the case where not every agent observing a stimulus is AWARE, there must be an UNAWARE agent observing stimuli, which on activation switches to the AWARE state with a constant probability p. This will happen in expected time $O(n/p) = O(n)$. Thus for the rest of the proof, we may assume every agent witnessing stimuli is already in the AWARE state with the witness flag set. Similarly, we may assume that at least one alert token has been generated as it would take at most $O(n)$ steps from a configuration without alert tokens.

We then bound the amount of time it takes for an alert token to reach an AWARE agent that is adjacent to an UNAWARE agent (which must exist as G is connected) To do this, we trace the movement of any one of the alert tokens as it gets passed around by the AWARE agents. When an agent carrying an alert token is activated, it picks a random neighbor to transfer the token to. Even though the algorithm does not pass the alert token in the event that the neighbor already holds an alert token, we may treat it as though the alert tokens have swapped positions for our analysis. The alert token follows a d_{max}-random walk, which has an expected hitting time of $O(n^2)$ movements of the alert token.

Thus, the expected number of iterations before some UNAWARE agent neighbors an agent with an alert token is $O(n^3)$. As there is a probability of at least $1/2$ of the UNAWARE agent being activated before the alert token moves away again, this gives us an expected time bound of $O(n^3)$ before a new AWARE agent is added. ◀

4 Reconfigurable topologies

We show that the same results hold if we allow some degree of reconfigurability of the edge set and relax the requirement that the graph must be connected. This gives us enough flexibility to implement a wider range of behaviors, like free aggregation or compression and dispersion in Section 5. When needed, we may refer to this as the *reconfigurable dynamic stimuli problem*, in order to clearly differentiate from the dynamic stimuli problem on static graphs that we considered in Section 3.

Instead of a static graph G, we now allow the edge set of a dynamic graph $\mathcal{G}$ to be locally modified over time. These reconfigurations can be initiated by the agents themselves or controlled by an adversary, and they can be randomized or deterministic, but we require some restrictions on what reconfigurations are allowed, and what information an algorithm carrying out these reconfigurations may have access to. This will result in a restricted class of dynamic graphs, but one that will be general enough to be applied to the problem of foraging that we describe in Section 5.

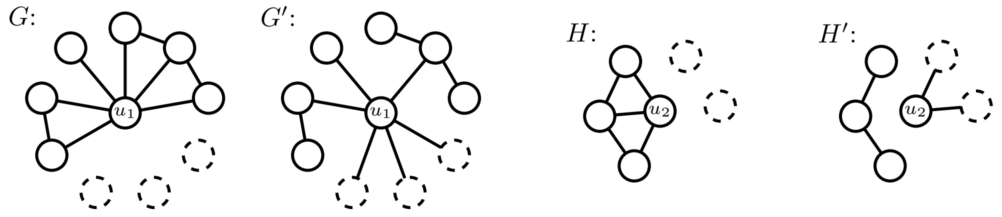


Figure 3 Two locally connected reconfigurations of the vertices u_1 (from G to G') and u_2 (from H to H') respectively, where only the AWARE neighbors of the reconfigured vertex are shown. Vertices with dashed outlines are newly introduced AWARE neighbors.

The basic primitive for (local) reconfiguration of our graph by an agent u is to replace the edges incident to vertex u with new edges. We call this a reconfiguration of vertex u and define local connectivity to formalize which reconfigurations are allowed.

▶ **Definition 11** (Locally Connected Reconfigurations). *For a configuration (G, θ), let G' be the resulting graph from the reconfiguration of some vertex $u \in V$, and where $N_{\mathcal{A}}(u)$ denotes the AWARE neighbors of u in G. We say that this reconfiguration (G', θ) is* locally connected *if u has at least one neighbor in G' that is AWARE and if for every pair of vertices v_1, v_2 in $N_{\mathcal{A}}(u)$, there is also a path from v_1 to v_2 in the induced subgraph $G'[N_{\mathcal{A}}(u) \cup \{u\}]$.*

Examples of locally connected reconfigurations are given in Figure 3.

In the reconfigurable dynamic stimuli problem, we want to be able to define reconfiguration behaviors for agents in the AWARE (without all-clear token) and UNAWARE states. To define what reconfigurations of an agent u are valid, we group our set of agent states into three subsets which we refer to as *behavior groups*. The three behavior groups are $\widehat{\mathcal{U}} := \{\mathcal{U}\}$,

$\widehat{\mathcal{M}} := \{\mathcal{A}_\emptyset, \mathcal{A}_{\{A\}}\}$ and $\widehat{\mathcal{I}} := \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}, \mathcal{A}_{\{C\}}\}$, which we call the *unaware*, *mobile* and *immobile* behavior groups We say that a locally connected reconfiguration of an agent u is *valid* if it is not allowed to reconfigure in the immobile behavior group, and if u is in the mobile behavior group, the reconfiguration of u must be locally connected (Definition 11). We show the following lemma, recalling that the state invariant appears in Definition 1:

▶ **Lemma 12.** *Let $G = (V, E)$ be a graph and let $G' = (V, E')$ be the graph resulting from a valid locally connnected reconfiguration of a vertex $u \in V$. If a state assignment θ of the agents V satisfies the state invariant on G, θ also satisfies the state invariant on G'.*

Proof. As locally connected reconfigurations of unaware agents do not affect the invariant and immobile agents cannot be reconfigured, it suffices to show that locally connected reconfigurations of mobile agents maintain the state invariant.

To show that the state invariant holds in G', we show that any mobile agent has a path to an immobile agent in G'. Consider any such mobile agent $v \neq u$. As the state invariant is satisfied in G, there exists a path in G over Aware vertices from v to an immobile agent w. If this path does not contain the agent u, v has a path to w in G'. On the other hand, if this path contains the agent u, consider the vertices u_1 and u_2 before and after u respectively in this path. By local connectivity, there must still be a path from u_1 to u_2 in the induced subgraph $G'[N_{\mathcal{A}}(u) \cup \{u\}]$, and so a path exists from v to w also in G'. It remains to check that u also has a path to an immobile agent in G'. Once again by local connectivity, u must have an Aware neighbor in G', which must have a path to an immobile agent in G'. ◀

In the reconfigurable version of the dynamic stimuli problem, instead of a static graph G, we consider a dynamic graph $\mathcal{G}$ where the snapshot G_t of $\mathcal{G}$ at each iteration t is generated by what we call a *(valid) reconfiguration adversary $\mathcal{X}$*. The reconfiguration adversary may store any amount of information known from previous iterations, which we will represent by the sequence $(X_0, X_1, \ldots)$. Then on each iteration t, it draws a new graph G_t and value X_t from the distribution $\mathcal{X}(X_{t-1}, \widehat{\theta}_{t-1})$, where the vectors $\widehat{\theta}_t : V \to \{\widehat{\mathcal{U}}, \widehat{\mathcal{M}}, \widehat{\mathcal{I}}\}$ denote the behavior groups of each agent at the end of each iteration t. Also, as an additional condition, the distribution $\mathcal{X}(X_{t-1}, \widehat{\theta}_{t-1})$ can only assign non-zero probabilities to graphs that can be obtained through some sequence of valid locally connected reconfigurations of the vertices of G_{t-1}. We are now ready to formally define *reconfigurable graphs*:

▶ **Definition 13** (Reconfigurable Graph)**.** *A dynamic graph $\mathcal{G}$ is called a* reconfigurable (dynamic) graph *if its evolution with time is given as a sequence of graphs $(G_0, G_1, \ldots)$ generated by a valid reconfiguration adversary $\mathcal{X}$.*

We note that the reconfiguration adversary can be deterministic or randomized (it can even be in control of the agents themselves), and is specifically defined to act based on the behavior group vectors $\widehat{\theta}_t : V \to \{\widehat{\mathcal{U}}, \widehat{\mathcal{M}}, \widehat{\mathcal{I}}\}$ and *not* the full state vector of the agents. We explicitly do not give the reconfiguration adversary access to full state information, as convergence time bounds require that the reconfiguration adversary of the graph be oblivious to the movements of alert tokens. As a special case, our results hold for any sequence of graphs $(G_0, G_1, G_2, \ldots)$ pre-determined by an oblivious adversary. An example of a valid reconfiguration adversary that takes full advantage of the generality of our definition can be seen in the Adaptive α-Compression Algorithm, an algorithm that we will later introduce to solve the problem of foraging.

In the static version of the problem, the graph is required to be connected to ensure that agents will always be able to communicate with each other. Without this requirement, we can imagine simple examples of graphs or graph sequences where no algorithm will work. In particular, if a set of agents that contains a stimulus never forms an edge to an agent outside of the set, there would be no way to transmit information about the existence of the stimulus to the nodes outside the set. However, as the foraging problem will require disconnections to some extent, we relax this requirement that each graph G_t is connected, and instead quantify how frequently UNAWARE state agents come into contact with AWARE state agents.

We say an UNAWARE agent is *active* if it is adjacent to an AWARE agent. One way to quantify how frequently agents become active is to divide the iterations into "batches" of bounded expected duration, with at least some amount of active agents in each batch. The random variables $(D_1, D_2, D_3 \ldots)$ denote the durations (in iterations) of these batches, and the random variables $(C_1, C_2, C_3, \ldots)$ denote the number of active agents in the respective batches. Definition 14 formalizes this notion.

▶ **Definition 14** (Recurring Sequences). *Let $\mathcal{X}$ be a fixed valid reconfiguration adversary. We say that this $\mathcal{X}$ is (U_D, U_C)-recurring (for $U_D \geq 1$ and $U_C \in (0,1)$) if for each possible starting iteration t and fixed behavior group $\widehat{\theta}$ with at least one unaware and one immobile agent, we have the following property: There exists sequences of random variables $(D_1, D_2, D_3 \ldots)$ and $(C_1, C_2, C_3 \ldots)$ where for each $k \in \{1, 2, 3, \ldots\}$,*

1. *C_k denotes the sum of the numbers of active agents under the behavior group $\widehat{\theta}$ over all iterations between iterations $t + \sum_{i=1}^{k-1} D_i$ and $t + \left(\sum_{i=1}^{k} D_i\right) - 1$.*
2. *$\mathbb{E}\left[D_k \mid D_1, D_2, \ldots D_{k-1}, C_1, C_2 \ldots C_{k-1}\right] \leq U_D$.*
3. *$\mathbb{E}\left[\left(1 - \frac{1}{n}\right)^{C_k} \mid D_1, D_2, \ldots D_{k-1}, C_1, C_2 \ldots C_{k-1}\right] \leq U_C$.*

We can then define a *(valid) reconfigurable graph (or sequence)* as one that is generated by a valid reconfiguration adversary $\mathcal{X}$ and is (U_D, U_C)-recurring for some $U_D \geq 1, U_C \in (0,1)$. This allows us to state our main results for reconfigurable graphs as Theorems 15 and 16. The theorems we have shown for the static version of the problem (Theorems 3 and 4) are special cases of these two theorems.

▶ **Theorem 15.** *Starting from any configuration satisfying the state invariant over a reconfigurable graph $\mathcal{G}$, if $|\mathcal{S}_T| = 0$, then all agents will reach and remain in the UNAWARE state in $O(n^2)$ expected iterations, after time T.*

▶ **Theorem 16.** *Starting from any configuration satisfying the state invariant over a reconfigurable graph $\mathcal{G}$, if $|\mathcal{S}_T| > 0$ and the reconfiguration adversary is (U_D, U_C)-recurring, then all agents will reach and stay in the AWARE state in $O\left(n^6 \log n + \frac{nU_D}{1-U_C}\right)$ expected iterations, after time T.*

In particular, if every graph G_t is connected, then the reconfiguration adversary is $\left(1, \left(1 - \frac{1}{n}\right)\right)$-recurring by setting $D_k = C_k = 1$ (as constant random variables) for all k, and we have the following corollary:

▶ **Corollary 17.** *Starting from any configuration satisfying the state invariant over a reconfigurable graph $\mathcal{G}$, if $|\mathcal{S}_T| > 0$ and every G_t is connected, then all agents will reach and stay in the AWARE state in $O(n^6 \log n)$ expected iterations, after time T.*

Obviously, if G is a static connected graph, it falls as a special case of the corollary; a tighter analysis allowed us to prove the $O(n^4)$ expected convergence bound in Theorem 4.

The following two lemmas, which are analogous to Lemmas 8 and 9 but for reconfigurable graphs, are sufficient to show Theorem 15 (the case for $|\mathcal{S}_T| = 0$). The proof of Lemma 18 is identical to the proof of Lemma 8, so we only show Lemma 19.

▶ **Lemma 18.** *We start from a configuration satisfying the state invariant over a reconfigurable graph $\mathcal{G}$ and assume that there are no more than s_{max} stimuli at any point of time. Then the expected number of steps before we reach a configuration with no residual components is at most $2n^2/(1 - p \cdot s_{max})$.*

▶ **Lemma 19.** *We start from a configuration satisfying the state invariant over a reconfigurable graph $\mathcal{G}$, and assume that no agent will be removed from the stimulus set from the current point on. If there are currently no residuals, then a residual cannot be generated.*

Proof. From the proof of Lemma 9, we know that state changes of agents do not generate a new residual. Valid reconfigurations of agents also cannot generate a new residual, as from Lemma 12, the state invariant always holds, so all components will always have an agent in state $\mathcal{A}_{\{C\}}$, $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$. Reconfigurations cannot change the fact that there will be no agent of state $\mathcal{A}_{\{C\}}$, and that all agents in states $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$ will be witnessing stimuli. ◀

The key result we will show is Lemma 20, a loose polynomial-time upper bound for the amount of time before the next agent switches to the AWARE state. This gives a polynomial time bound for all agents switching to the AWARE state when $|\mathcal{S}_T| > 0$, implying Theorem 16.

▶ **Lemma 20.** *We start from a configuration satisfying the state invariant over a (U_D, U_C)-recurring reconfigurable graph $\mathcal{G}$, and assume that there are no residuals, the stimulus set is nonempty, and no agent will be removed from the stimulus set from the current point on. Then the expected number of iterations before the next agent switches from the UNAWARE to the AWARE state is at most $O\left(n^5 \log n + U_D/(1 - U_C)\right)$.*

Proof. As proved in Lemma 10, any UNAWARE agent witnessing stimuli will become aware in expected $O\left(n/p\right)$ time, so for the rest of the proof, we may assume that every agent witnessing stimuli is already in the AWARE state with the witness flag set. From then on however, our proof strategy differs from Lemma 10 to account for the reconfigurability of $\mathcal{G}$. We show a loose time bound for reconfigurable graphs in two parts: first we bound the time it takes for all agents to hold an alert token, followed by a bound on the time it takes for an UNAWARE agent to be activated while adjacent to an AWARE agent holding an alert token.

We first bound the amount of time it takes for all AWARE state agents to hold alert tokens. As an agent can only hold one alert token at a time, a new alert token can only be generated when the agent witnessing a stimulus does not currently hold an alert token. Assume that there is at least one AWARE state agent that does not hold an alert token, and mark this agent. We bound how long it takes in expectation for the mark to land on an agent witnessing the stimulus, as a new alert token can then be generated following that with constant probability. Suppose that u is the marked agent and that the next agent v to be activated is a neighbor of u. If v has an alert token and v randomly chooses u as its outgoing neighbor (as per the algorithm), then v transfers its alert token to u and receives the mark from u. Otherwise, for the sake of our analysis, we still have v pick an outgoing neighbor at random and receive the mark if the chosen neighbor is u.

The mark moving in this manner is equivalent to following a d_{max}-random walk over the subgraphs induced by the AWARE agents on each iteration. It is known that the expected hitting time of the d_{max}-random walk on a connected evolving graph controlled by an *oblivious adversary* is $O(n^3 \log n)$ [3, 16]. This polynomial time bound is notable as there are connected evolving graphs where the simple random walk admits exponential hitting times in the worst case [3]. In our model, this corresponds to $O(n^4 \log n)$ iterations in expectation to generate a new alert token, which gives an upper bound of $O(n^5 \log n)$ iterations in expectation before all agents carry alert tokens. This is clearly a loose bound, which has not been optimized for clarity of explanation. Two complications however arise when applying this result: the requirement that the adversary controlling the dynamic graph be oblivious and the requirement that the dynamic graph remain connected.

We first address the requirement for the adversary to be oblivious. Unlike the proof of Lemma 10, the mark now moves over the sequence of induced subgraphs $G_{T_{\text{start}}+1}, G_{T_{\text{start}}+2}, \ldots$, where T_{start} denotes the last iteration an UNAWARE state agent switched to the AWARE state. On each iteration $t \geq T_{\text{start}}$, G_{t+1} is drawn from a distribution $\mathcal{X}(X_t, \widehat{\theta}_t) = \mathcal{X}(X_t, \widehat{\theta}_{T_{\text{start}}})$, as $\widehat{\theta}_t = \widehat{\theta}_{T_{\text{start}}}$ for all $t \geq T_{\text{start}}$ (note that as long as no new UNAWARE agent switches to the AWARE state, as there are no residuals and the set of stimuli $\mathcal{S}_T$ is no longer changing, the behavior group vectors $\widehat{\theta}_{T_{\text{start}}}, \widehat{\theta}_{T_{\text{start}}+1}, \ldots, \widehat{\theta}_t$ will not have changed since T_{start}). Thus conditioned on everything that has happened on iterations up to and including T_{start}, the random sequence describing the movements of the mark is independent of the random sequence describing the reconfiguration behavior of the graph.

We next address the connectivity requirement of the dynamic graph. Denote by A the set of AWARE state agents given by $\widehat{\theta}_{T_{\text{start}}}$. The sequence of induced subgraphs $G_t[A], t \geq T_{\text{start}} + 1$, to which we would like to apply the result of [3, 16], in general may not be connected. This problem has a relatively straightforward solution: as the state invariant (Definition 1) holds, each connected component of $G_t[A]$ must contain at least one immobile agent (each of which witnesses a stimulus as there are no residuals). Adding an edge between every pair of immobile agents in $G_t[A]$ for each t gives a sequence of connected graphs H_t. This allows us to see the movements of the mark as a d_{max}-random walk over the sequence of connected graphs H_t.

Now that all AWARE agents have alert tokens, all that remains is to activate an UNAWARE agent neighboring an AWARE agent (otherwise known as an active agent) and convert it to an AWARE agent, consuming the alert token held by said neighbor. Denote by T_{full} the first iteration where every AWARE agent carries an alert token. As the reconfiguration adversary is (U_D, U_C)-recurring, the iterations following T_{full} may be divided into intervals with lengths denoted by the random variables $D_1, D_2, D_3, \ldots$, where the total numbers of active agents within the respective intervals are denoted by the random variables $C_1, C_2, C_3, \ldots$.

A new AWARE agent is added when an active UNAWARE agent is activated. The probability of adding a new AWARE agent on a given iteration with k active agents is thus $\frac{k}{n}$, as each of the n agents are activated with equal probability. Thus, if we denote the number of active agents on an iteration starting from T_{full} by the random sequence $K_1, K_2, K_3, \ldots$, we get the following expression for the number of iterations starting from T_{full} before a new AWARE agent is added, which we denote by X:

$$\begin{aligned}
\mathbb{E}[X|K_1, K_2, K_3, \ldots] &= \sum_{x=0}^{\infty} Pr\left(X > x | K_1, K_2, K_3, \ldots\right) \\
&= 1 + \sum_{x=1}^{\infty} \left(1 - \frac{K_1}{n}\right)\left(1 - \frac{K_2}{n}\right) \cdots \left(1 - \frac{K_x}{n}\right) \\
&\leq 1 + \sum_{x=1}^{\infty} \left(1 - \frac{1}{n}\right)^{K_1} \left(1 - \frac{1}{n}\right)^{K_2} \cdots \left(1 - \frac{1}{n}\right)^{K_x} \\
&= 1 + \sum_{x=1}^{\infty} y^{\left(\sum_{i=1}^{x} K_i\right)},
\end{aligned}$$

where we define $y := \left(1 - \frac{1}{n}\right) \in (0,1)$. Thus,

$$\begin{aligned}
&\mathbb{E}[X|K_1, K_2, K_3, \ldots] \\
&\quad \leq \; 1 + \underbrace{y^{K_1} + y^{K_1+K_2} + \ldots + y^{\sum_{i=1}^{D_1} K_i}}_{D_1 \text{ terms}} + \underbrace{y^{C_1 + K_{D_1+1}} + \ldots + y^{C_1 + \sum_{i=D_1+1}^{D_1+D_2} K_i}}_{D_2 \text{ terms}} + \ldots \\
&\quad \leq \; 1 + (D_1 - 1) + y^{C_1} + y^{C_1}(D_2 - 1) + y^{C_1+C_2} + y^{C_1+C_2}(D_3 - 1) + y^{C_1+C_2+C_3} + \ldots \\
&\quad = \; D_1 + D_2 \cdot y^{C_1} + D_3 \cdot y^{C_1+C_2} + \ldots,
\end{aligned}$$

Via the law of total expectation, we have

$$\begin{aligned}
\mathbb{E}[X] &\leq \sum_{i=1}^{\infty} \mathbb{E}[D_i y^{\sum_{j=1}^{i-1} C_j}] \\
&= \sum_{i=1}^{\infty} \mathbb{E}\left[y^{\sum_{j=1}^{i-1} C_j} \mathbb{E}[D_i | C_1, C_2, \ldots, C_{i-1}]\right] \\
&\leq \sum_{i=1}^{\infty} U_D \mathbb{E}\left[y^{\sum_{j=1}^{i-2} C_j} \mathbb{E}[y^{C_{i-1}} | C_1, C_2, \ldots, C_{i-2}]\right] \\
&\leq \ldots \leq \sum_{i=1}^{\infty} U_D (U_C)^{i-1} = \frac{U_D}{1 - U_C}.
\end{aligned}$$

Combining the two phases, we have an expected time bound of $O\left(n^5 \log n + \frac{U_D}{1-U_C}\right)$ before a new Aware agent is added. ◀

5 Foraging via self-induced phase changes

Recall that in *the foraging problem*, we have "ants" (agents) that may initially be searching for "food" (stimuli, which can be any resource in the environment, like an energy source). Once a food source is found, the ants that have learned about the food source start informing other ants, allowing them to switch their behaviors from the *search mode* to the *gather mode*, leading them to start gathering around the food to consume it. Once the source is depleted, the ants closer to the depleted source start a broadcast wave, gradually informing other ants that they should restart the search phase again by individually switching their states. The foraging problem is very general and has several fundamental application domains, including search-and-rescue operations in swarms of nano- or micro-robots; health applications (e.g., a collective of nano-sensors that could search for, identify, and gather around a foreign body to isolate or consume it, then resume searching, etc.); and finding and consuming/deactivating hazards in a nuclear reactor or a minefield.

Our model for foraging is based on the *geometric amoebot model* for programmable matter, first introduced in [17] (see also [15]). We have n anonymous agents occupying distinct sites on a $\sqrt{N} \times \sqrt{N}$ region of the triangular lattice with periodic boundary conditions, where $n \leq N$. These agents have constant-size memory, and have no global orientation or any other global information beyond a common chirality. Agents are activated with individual Poisson clocks (with uniform rates), which is equivalent to assuming a *random sequential scheduler*. Upon activation, an agent may move to an adjacent unoccupied lattice site and change its state and that of its neighbors, operating under similar constraints to the dynamic stimuli problem.[3] An agent may only communicate with agents occupying adjacent sites of the lattice. Food sources may be placed on any site of the lattice, removed, or shifted around at arbitrary times, possibly adversarially, and an agent can only observe the presence of the food source while occupying the lattice site containing it.

At any point of time, there are two main states an agent can be in, which, at the macro-level, are to induce the collective to enter the *search* or *gather* modes respectively. When in *search mode*, agents move around in a process akin to a simple exclusion process, where they perform a random walk while avoiding two agents occupying the same site. Agents enter the *gather mode* when food is found and this information is propagated in the system, consequently resulting in the system compressing around the food (Figure 4).

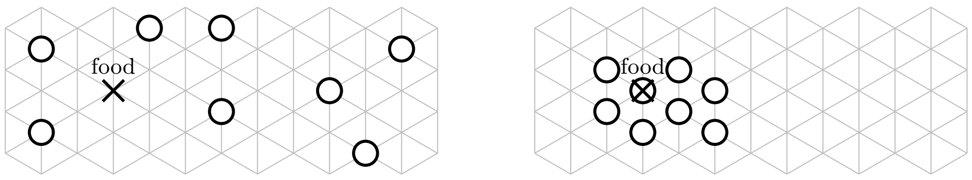


Figure 4 In the diagram on the left, a food source is placed on a lattice site. The diagram on the right illustrates a desired configuration, where all agents have gathered in a low perimeter configuration around the food source. If the food source is later removed, the agents should once again disperse, returning to a configuration like the figure on the left.

In the nonadaptive setting, Cannon *et al.* [7] designed a rigorous compression/expansion algorithm for agents in the amoebot model that remain simply connected throughout execution, where a single parameter λ determines a system-wide phase: A small λ, namely $\lambda < 2.17$, provably corresponds to the search mode, which is desirable to search for food, while large λ, namely $\lambda > 2+\sqrt{2}$, corresponds to the gather mode, desirable when food has been discovered. Likewise, Li *et al.* [30] show a very similar bifurcation based on a bias parameter λ in the setting when the agents are allowed to disconnect and disperse throughout the lattice. Our goal here is to perform a system-wide adjustment in the bias parameters when one or more agents notice the presence or depletion of food to induce the appropriate global coordination to provably transition the collective between macro-modes when required. Informally, one can imagine individual agents adjusting their λ parameter to be high when they are fed, encouraging compression around the food, and making λ small when they are hungry, promoting the search for more food. A configuration is called *α-compressed* if the perimeter (measured by the length of the closed walk around its boundary edges) is at most $\alpha\, p_{\min}(n)$, for some constant $\alpha > 1$, where $p_{\min}(n)$ denotes the minimum possible perimeter of a connected system with n agents, which is the desired outcome of the gather mode.

5.1 Adaptive α-compression

We present the first rigorous local distributed algorithm for the foraging problem: The *Adaptive α-Compression* algorithm is based on the stochastic compression algorithm of [7], addressing a geometric application of the dynamic stimuli problem, where the Aware state represents the "gather" mode, and the Unaware state represents the "search" mode. An agent witnesses a stimuli if it occupies the same lattice site as the food (stimulus) source.

[3] Assuming that amoebots can write directly to their neighbors' memories is common in the literature under the sequential scheduler, e.g., see [13, 18]; it would be interesting to port the model and algorithms in this paper to the message-passing canonical amoebot model [15] (see Section 7).

Algorithm 2 Adaptive α-Compression

```
1: procedure ADAPTIVE-ALPHA-COMPRESSION($u$)
2:     $q \leftarrow$ Random number in $[0,1]$
3:     $u.isWitness \leftarrow$TRUE if $u$ observes the food source, else $u.isWitness \leftarrow$FALSE
4:     if $q \leq \frac{1}{2}$ then                    ▷ With probability $\frac{1}{2}$, make a state update
5:         ADAPTIVE-STIMULI-ALGORITHM(u)
6:     else                                           ▷ With probability $\frac{1}{2}$, make a move
7:         if $u.isWitness$ or $u.state \in \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}, \mathcal{A}_{\{C\}}\}$ then    ▷ immobile agent
8:             Do nothing
9:         else if $u.state \in \{\mathcal{A}_{\emptyset}, \mathcal{A}_{\{A\}}\}$ then                 ▷ mobile agent
10:            EXECUTE-GATHER(u)
11:        else if $u.state = \mathcal{U}$ then                                  ▷ unaware agent
12:            EXECUTE-SEARCH(u)

1: procedure EXECUTE-GATHER($u$)
2:     $d \leftarrow$ Random direction in $\{0,1,2,3,4,5\}$
3:     $\ell \leftarrow$ Current position of $u$
4:     $\ell' \leftarrow$ Neighboring lattice site of $u$ in direction $d$
5:     if Moving $u$ from $\ell$ to $\ell'$ is a valid compression move (Definition 21) then
6:         $p \leftarrow$ Random number in $[0,1]$
7:         $d(u) \leftarrow$ number of neighboring agents of $u$ if $u$ were at position $\ell$
8:         $d'(u) \leftarrow$ number of neighboring agents of $u$ if $u$ were at position $\ell'$
9:         if $p \leq \lambda^{d'(u)-d(u)}$ then
10:            Move $u$ to position $\ell'$          ▷ Movements reconfigure the adjacency graph

1: procedure EXECUTE-SEARCH($u$)
2:     $d \leftarrow$ Random direction in $\{0,1,2,3,4,5\}$
3:     $\ell' \leftarrow$ Neighboring lattice site of $u$ in direction $d$
4:     if Moving $\ell'$ is an unoccupied lattice site then
5:         Move $u$ to position $\ell'$              ▷ Movements reconfigure the adjacency graph
```

The underlying dynamic graph used by the Adaptive Stimuli Algorithm is given by the adjacency graph of the agents, where two agents share an edge on the graph if they occupy adjacent sites of the lattice. As agents move around to implement behaviors like gathering and searching, their neighbor sets will change. The movement of agents thus reconfigures and oftentimes even disconnects our graph.

In this algorithm, agents in the unaware (search) behavior group execute movements akin to a simple exclusion process (EXECUTE-SEARCH) while agents in the mobile (gather) behavior group execute moves of the compression algorithm (EXECUTE-GATHER) in [7].[4] We focus on the compression algorithm run by the AWARE agents. In a simple exclusion process, a selected agent picks a direction at random, and moves in that direction if and only if the immediate neighboring site in that direction is unoccupied. In the compression algorithm [7] on the other hand, a selected AWARE agent first picks a direction at random to move in, and if this move is a valid compression move (according to Definition 21), the agent moves to the chosen position with the probability given in Definition 22. With a (far-from-trivial)

[4] Following the description of these procedures in [7], we describe the movements of particles (agents) in a combined expansion and contraction; those can be decoupled into two separate activations of the agent, since the amoebot model allows at most one expansion or contraction per activation.

modification of the analysis in [7] to account for the stationary witness agent, we show that in the case of a single food source, the "gather" movements allow the agents to form a low perimeter cluster around the food. We present the following definitions, adapted from [7] to the set of AWARE agents running the compression algorithm:

▶ **Definition 21** (Valid Compression Moves [7]). *Denote by $N_{\mathcal{A}}(\ell)$ and $N_{\mathcal{A}}(\ell')$ the sets of* AWARE *neighbors of ℓ and ℓ' respectively and $N_{\mathcal{A}}(\ell \cup \ell') := N_{\mathcal{A}}(\ell) \cup N_{\mathcal{A}}(\ell') \setminus \{\ell, \ell'\}$. Consider the following two properties:*

Property 1: $|N_{\mathcal{A}}(\ell) \cap N_{\mathcal{A}}(\ell')| \geq 1$ and every agent in $N_{\mathcal{A}}(\ell \cup \ell')$ is connected to an agent in $N_{\mathcal{A}}(\ell) \cap N_{\mathcal{A}}(\ell')$ through $N_{\mathcal{A}}(\ell \cup \ell')$.

Property 2: $|N_{\mathcal{A}}(\ell) \cap N_{\mathcal{A}}(\ell')| = 0$, ℓ and ℓ' each have at least one neighbor, all agents in $N_{\mathcal{A}}(\ell) \setminus \{\ell'\}$ are connected by paths within the set, and all agents in $N_{\mathcal{A}}(\ell') \setminus \{\ell\}$ are connected by paths within the set.

We say the move from ℓ to ℓ' is a valid compression move *if it satisfies both properties, and $N_{\mathcal{A}}(\ell)$ contains fewer than five aware state agents.*

▶ **Definition 22** (Transition probabilities [7]). *Fix $\lambda > 2 + \sqrt{2}$, as sufficient for α-compression. An agent u transitions through a valid movement with Metropolis-Hastings [34] acceptance probability* $\min\{1, \lambda^{e(\sigma')-e(\sigma)}\}$*, where σ and σ' are the configurations before and after the movement, and $e(\cdot)$ represents the number of edges between* AWARE *state agents in the configuration.*

Note that even though $e(\cdot)$ is a global property, the difference $e(\sigma') - e(\sigma)$ can be computed locally (within two hops in the lattice, or through expansions in the amoebot model [14, 15]), as it is just the change in the number of AWARE neighbors of u before and after its movement.

The condition for valid compression moves is notable as it keeps a component of aware agents containing the agent witnessing the stimulus simply connected (connected and hole-free), which is crucial to the proof in [7] that a low perimeter configuration, in our case around the food source, will be obtained in the long term. We also show that these valid compression moves are locally connected (as per Definition 11), a sufficient condition for the reconfigurable dynamic stimuli problem to apply.

5.2 Correctness

We now prove the various claims underlying the foraging algorithm. We first require that the Markov chain representing the compression moves (EXECUTE-GATHER) is connected. The proof builds upon the ergodicity argument in [7], but the addition of a single stationary agent, the witness, in our context makes it significantly more complex. The Markov Chain is trivially aperiodic, since there is a nonzero probability that an agent may not move (in EXECUTEGATHER) when activated; Lemma 23 shows that it is also irreducible. Since the irreducibility proof may be of independent interest and is rather involved, we present it as a stand-alone proof in Section 6.

▶ **Lemma 23** (Irreducibility). *Consider connected configurations of agents on a triangular lattice with a single agent v that cannot move. There exists a sequence of valid compression moves that transforms any connected configuration of agents into any simply connected configuration of the same agents while keeping v stationary.*

We may now state our main results, which verify the correctness of Adaptive α-Compression.

▶ **Theorem 24.** *If no food source has been identified for sufficiently long, then within an expected $O(n^2)$ steps, all agents will reach and remain in the* Unaware *state and will converge to the uniform distribution of nonoverlapping lattice positions.*

▶ **Theorem 25.** *If at least one food source exists and remains in place for long enough, then within $O(n^6 \log n + N^2 n^2)$ steps in expectation, all agents will reach and remain in the* Aware *state, and each component of* Aware *agents will contain a food source. In addition, if there is only one food source, the agents will converge to a configuration with a single α-compressed component around the food, for any constant $\alpha > 1$, with all but an exponentially small probability, if the lattice region size N is sufficiently large.*

The Adaptive α-Compression Algorithm fits the requirements of the reconfigurable dynamic stimuli model. In particular, the information X_t available to the reconfiguration adversary corresponds to the configuration of the lattice, and the graph G_t represents the adjacency of agents on the lattice at that time t. To show that a sequence of valid Aware agent movements in the Adaptive α-Compression Algorithm, which determine the configurations of $G_1, G_2, \ldots$ can be modeled via a valid reconfiguration adversary $\mathcal{X}$, we need to show that the reconfigurations resulting from the Execute-Gather procedure must be locally connected.

▶ **Lemma 26.** *The movement behavior of Adaptive α-Compression is locally connected.*

Proof. We only need to show that the Execute-Gather procedure maintains local connectivity (Definition 11). This is true as when reconfiguring an Aware agent u with Aware neighbor set $N_{\mathcal{A}}(u)$, we only allow valid compression moves (Definition 21) to be made. In the case of Property 1, all agents in $N_{\mathcal{A}}(u)$ will still have paths to u in G' (the resulting graph of the reconfiguration) through $N_{\mathcal{A}}(u) \cap N'_{\mathcal{A}}(u)$ where $N'_{\mathcal{A}}(u)$ is the Aware neighbor set of u in G'. In the case of Property 2, all agents in $N_{\mathcal{A}}(u)$ will still have paths to each other within $G'[N_{\mathcal{A}}(u)]$, despite no longer having local paths to u. The agent u will have at least one Aware state neighbor after the move as this is a requirement of Property 2. ◀

As this is an instance of the reconfigurable dynamic stimuli problem, Theorem 24 follows immediately from Theorem 15. To show Theorem 25, however, we need to show polynomial recurring rates by arguing that the Unaware agents following a simple exclusion process regularly come into contact with the clusters of Aware state agents around the food sources.

▶ **Lemma 27.** *The movement behavior defined in the Adaptive α-Compression Algorithm is (U_D, U_C)-recurring with $U_D = 2N^2 n + \frac{n}{2} + 1$ and $U_C = \frac{2}{3}$.*

Proof. As required by Definition 14, We start from an arbitrary system configuration and assume an arbitrary initial behavior group vector $\widehat{\theta}$ that will remain fixed for this analysis. This behavior group vector will also be assumed to contain at least one Unaware and at least one immobile agent. In adaptive α-compression, as an agent's movement depends only on the current configuration of the lattice and $\widehat{\theta}$, for convenience of notation and without loss of generality, we assume that we are on the starting iteration 0.

In each iteration, the Adaptive α-Compression Algorithm randomly chooses between executing a step of the dynamic stimuli problem and executing a movement on the lattice, where there may be any number of movements between steps of the dynamic stimuli problem. Throughout this proof, to distinguish between these two possibilities, a *dynamic stimuli step* will be used to refer to an execution of the dynamic stimuli algorithm, while a *movement step* will be used to refer to to an execution of the movement algorithm.

To show that the reconfiguration adversary is recurring, the random sequences $(D_1, D_2, \ldots)$ and $(C_1, C_2, \ldots)$ as required by Definition 14 need to be defined—these random sequences partition the iterations into *"batches,"* where the i^{th} batch would take D_i dynamic stimuli steps and would see C_i active agents (an active agent is an UNAWARE agent that neighbors an AWARE agent) over its duration. Suppose that batch $i-1$ ended on the T^{th} dynamic stimuli step (if $i = 1$, then $T = 0$) which occurs on iteration t_0^i. To define when the next batch ends, we consider two important iterations of the algorithm in batch i:

- t_1^i: The first iteration after dynamic stimuli step T where an agent movement puts an UNAWARE agent u next to an AWARE agent v.
- t_2^i: The first iteration after t_1^i where u or v are selected again in a movement step (even if no action ends up being taken in the movement step).

The batch i thus refers to the dynamic stimuli steps starting from $T+1$ and ending on the first dynamic stimuli step T' following t_2^i (the following batch $i+1$ then starts on dynamic stimuli step $T'+1$).

To upper bound the expected number of movement steps required to reach t_1^i, we note that the movements of the agents in the UNAWARE state follow a simple exclusion process. For the sake of the analysis, we treat situations where a movement of an UNAWARE agent is rejected due to an obstruction by another UNAWARE agent as a "swap" of the positions of the two agents. We can then analyze the hitting time of a simple exclusion process as that of an independent simple random walk over the lattice region. If AWARE agents did not exist, as our triangular lattice region (with periodic boundary conditions) is a regular graph, a simple worst-case upper bound for how long it takes in expectation before an UNAWARE agent reaches the location of an immobile agent is $2N^2$ movements of the agent [32], which translates to $2N^2n$ movement steps in expectation. The UNAWARE agent must come into contact with an AWARE agent (which can be mobile or immobile) before this happens, so $2N^2n$ is an upper bound for the expected number of movement steps between t_0^i and t_1^i.

We next give an upper bound for the expected number of movement steps between t_1^i and t_2^i. The probability of selecting u or v in a movement step is exactly $\frac{2}{n}$, so the expected value of a geometric distribution gives us an upper bound of $\frac{n}{2}+1$ movement steps in expectation. As the number of movement steps is equal to the number of dynamic stimuli steps in batch i in expectation, and this applies from any starting configuration, this gives us a uniform upper bound $\mathbb{E}[D_i | D_1, D_2, \ldots D_{i-1}, C_1, C_2 \ldots C_{i-1}] \leq 2N^2n + \frac{n}{2} + 1$ for all batches $i \in \{1, 2, \ldots\}$.

To lower bound C_i, we observe that there will be at least one active agent neighboring an AWARE agent, on any dynamic stimuli step occurring between iterations t_1^i and t_2^i, since neither u nor v will be selected in a movement step before t_2^i. Thus the number of dynamic stimuli steps between t_1^i and t_2^i lower bounds C_i, the sum of the numbers of active agents over the dynamic stimuli steps of batch i. For any two agents u' and v', let $Y_{u',v'}$ be a random variable denoting the number of dynamic stimuli steps that would pass before the algorithm selects u' or v' in a movement step. As there are $X \sim \text{Geom}(\frac{1}{2})$ movement steps in between any two dynamic stimuli steps and the probability of the algorithm selecting u' or v' on any movement step is $\frac{2}{n}$, the random variable $Y_{u',v'}$ is geometrically distributed with a success probability $p_{Y_{u',v'}}$ where:

$$\begin{aligned} p_{Y_{u',v'}} &= \sum_{j=0}^{\infty} Pr(X=j) \cdot Pr(\text{algorithm selects } u' \text{ or } v' \text{ within } j \text{ movement steps}) \\ &= \sum_{j=0}^{\infty} \left(\frac{1}{2}\right)^{j+1} \left(1 - \left(1 - \frac{2}{n}\right)^j\right) = 1 - \frac{1}{2}\sum_{j=0}^{\infty}\left(\frac{1-2/n}{2}\right)^j = \frac{2}{n}\left(\frac{1}{1+2/n}\right). \end{aligned}$$

We can thus say that C_i (conditioned on past D_k, C_k) stochastically dominates $Y_{u,v}$, and because $(1-\frac{1}{n})^x$ is a decreasing function of x, we have:

$$\begin{aligned}\mathbb{E}\left[\left(1-\frac{1}{n}\right)^{C_i}\middle| D_1, D_2, \ldots D_{i-1}, C_1, C_2 \ldots C_{i-1}\right] &\leq \mathbb{E}\left[\left(1-\frac{p}{n}\right)^{Y_{u,v}}\right] \\ &= \sum_{y=0}^{\infty}\left(1-\frac{1}{n}\right)^{y} p_{Y_{u,v}}\left(1-p_{Y_{u,v}}\right)^{y} \\ &= p_{Y_{u,v}}\frac{1}{1-(1-1/n)(1-p_{Y_{u,v}})} = \frac{2}{3}. \qquad \blacktriangleleft\end{aligned}$$

We are now ready to present the proof of Theorem 25:

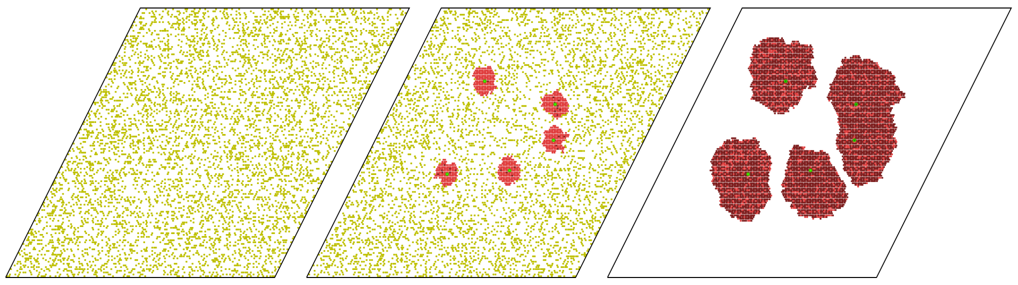
Proof of Theorem 25. We first show that if at least one food source exists and remains in place for long enough, then within $O(n^6 \log n + N^2 n^2)$ steps in expectation, all agents will reach and remain in the Aware state, and each component of Aware agents will contain a food source. We start from the first iteration beyond which no additional changes in the positions (or existence) of the food sources occur. We first show that if there is at least one food source, it will be found. As long as no food source has been found, there will be no witnesses to stimuli, so every agent will return to the Unaware state by Theorem 15. Agents in the Unaware state move randomly following a simple exclusion process. Using the hitting time of a simple random walk on a regular graph (the triangular lattice region, with periodic boundary conditions) of N sites, we have a simple upper bound of $O(N^2 n)$ iterations in expectation before some agent finds a food source and becomes a witness.

From then on, there will be at least one stimulus, and the stimulus set can only be augmented, not reduced, as the other agents potentially find additional food sources, and since the agents witnessing food sources are no longer allowed to move. As by Lemma 27 agent movement behaviors are (U_D, U_C)-recurring with with $\frac{U_D}{1-U_C} = O(N^2 n)$, we can apply Theorem 16, which yields a polynomial bound of $O(n^6 \log n + N^2 n^2)$ on the expected number of iterations until all agents have switched to the Aware state with no residuals. Additionally, due to the state invariant, once there are no residuals, every component of Aware agents will contain at least one stimulus (food source). This gives us the first part of Theorem 25.

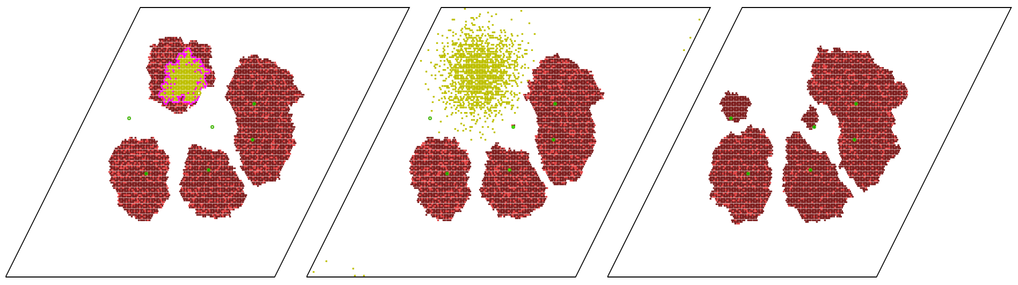
The second half of Theorem 25 states that a low perimeter (α-compressed) configuration is always achievable in the case of a single food source. As the Markov chain representing the compression moves is irreducible (Lemma 23), the results of [7] guarantee that for any $\alpha > 1$, there exists a sufficiently large constant λ such that at stationarity, the perimeter of the cluster is at most α times its minimum possible perimeter with high probability. ◀

5.3 Multiple food sources and simulations

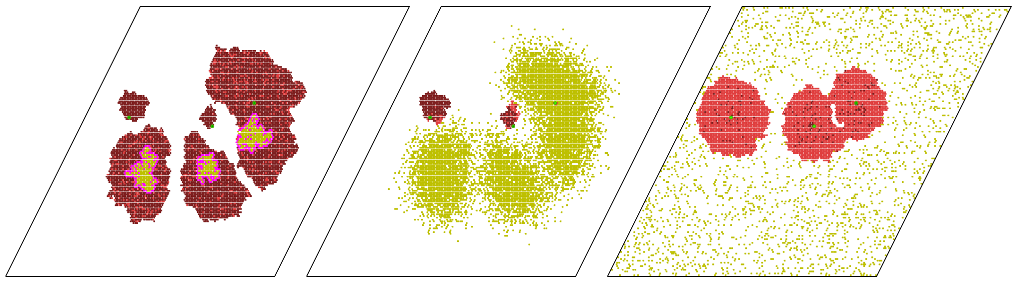
We now consider the more general case where there can be multiple food sources. If more than one food sources exist and remain stable for sufficiently long, our algorithm will correctly switch all the agents to the Aware state as needed, with each agent in a component containing at least one food source, per Theorem 25. However, we cannot guarantee that the set of agents will converge to α-compressed configurations due to certain pathological cases where the immobile food sources can become obstacles constricting some of the clusters, preventing the α-compression algorithm from reaching compressed states. This seems to only be an issue when food sources are sufficiently close and we believe that when the food sources are reasonably separated, the ensemble will form one or more α-compressed components. The irreducibility proof in the presence of a single immobile agent (presented in Section 6) is an



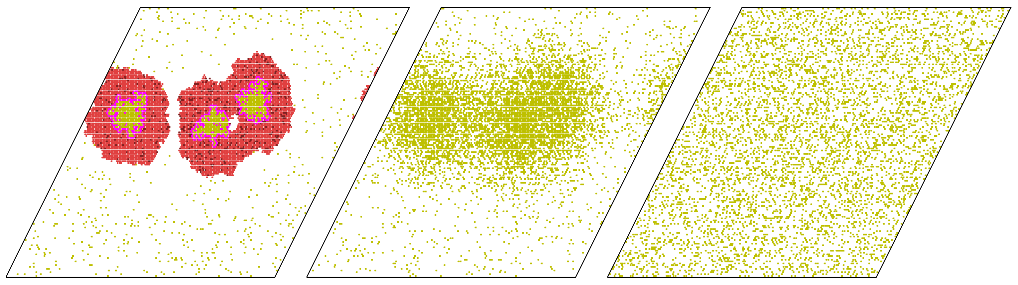
(a) All agents initially unaware. Five food sources added, all agents eventually switch to the Aware state.



(b) One food source removed, two new added. A cluster disperses and agents rejoin other clusters.



(c) Three food sources removed, clusters disperse and gather around the remaining food sources.



(d) All food sources removed, agents disperse and converge to a uniform distribution over the lattice.

Figure 5 Simulation of Adaptive α-Compression with multiple food sources. The images are in chronological order. Unaware agents are yellow, Aware agents are red (darker red if they hold an alert token), agents with the all-clear token are purple, and food sources are green.

extension of the proof of irreducibility of the Markov chain for compression without immobile agents given in [7]; we believe it would possible to further extend the proof to include cases with multiple food sources under specific conditions.

In Figure 5, we demonstrate a simulation of the Adaptive α-Compression algorithm with multiple food sources and 5625 agents, in a 150×150 region of the triangular lattice with periodic boundary conditions. At initialization (iteration 0), all agents are in the Unaware state and are placed uniformly at random. Multiple food sources (stimuli) are then manually placed and moved around to illustrate the gather and search phases. This simulation is shown as a sequence of 12 images in chronological order.

In addition, we ran experiments on a $\sqrt{N} \times \sqrt{N}$ grid with a single food source for different values of N (between 100 and 22500) and various particle densities (between 0.1 and 1). These experiments suggest a running time of between $O(N^2)$ and $O(N^3)$ activations in expectation for all agents to switch to the Aware state from a configuration where all agents are initially Unaware, and between $O(N)$ and $O(N^2)$ activations in expectation for all agents to return to the Unaware state when a food source is removed from a configuration where all agents are initially Aware. This is significantly faster than our loose polynomial upper bounds for the algorithm.

6 Ergodicity of the Markov chain for compression

We now conclude by presenting the proof of Lemma 23, showing that the Markov chain for compression is irreducible (and thus ergodic, since it is trivially aperiodic, with self-loops) in the presence of an immobile agent on the food source, as Algorithm 2 relies on this result. This proof may also be of independent interest, with applications to other problems or domains. The proof of irreducibility without an immobile agent given in Cannon et al. [7] was already fairly involved and the addition of the immobile agent requires a more careful and subtle analysis.

The main strategy in the proof is to treat the immobile agent on the food source as the “center” of the configuration, and consider the lines extending from the center in each of the six possible directions. These lines, which we call “spines”, divide the lattice into six regions. The sequence of moves described in [7] is then modified to operate within one of these regions, with limited side effects on the two regions counterclockwise from this region. We call this sequence of moves a “comb”, and show that there is a sequence of comb operations that can be applied to the configuration, repeatedly going round the six regions in counterclockwise order, until the resulting configuration is a single long line. We then observe that any valid compression move transforming a hole-free configuration to a hole-free configuration is also valid in the reverse direction, giving us the statement of the lemma.

We will treat the immobile agent on the food source as the “center” of the configuration. From the center, as seen in Figure 6, there are six directions one can move in a straight line on the triangular lattice: up, down, up-left, down-left, up-right, down-right. We call these six straight lines of agents extending from the immobile agent *spines*. We refer to agents on the spines as *spine agents*, and agents not on spines as *non-spine agents*. We similarly use the names spine and non-spine locations to refer to sites on the triangular lattices.

On each spine, out of its spine agents that have adjacent non-spine agents, we call the furthest out such agent from the immobile agent the anchor agent of the spine (these are highlighted in red in Figure 6). The *distance* of a spine location from the center refers to its shortest-path distance (which would be along the spine) to the immobile agent on the triangular lattice. For each integer $r \geq 1$, the hexagon of radius r refers to the regular hexagon with corners defined by the six spine locations of distance r from the center. The distance of a non-spine location from the center would then be the radius of the smallest such hexagon it is contained within. An important concept that we will use in the proof is the length of a spine, defined to be the distance of its anchor agent to the center. If it has no anchor agent, the length of the spine is 0.

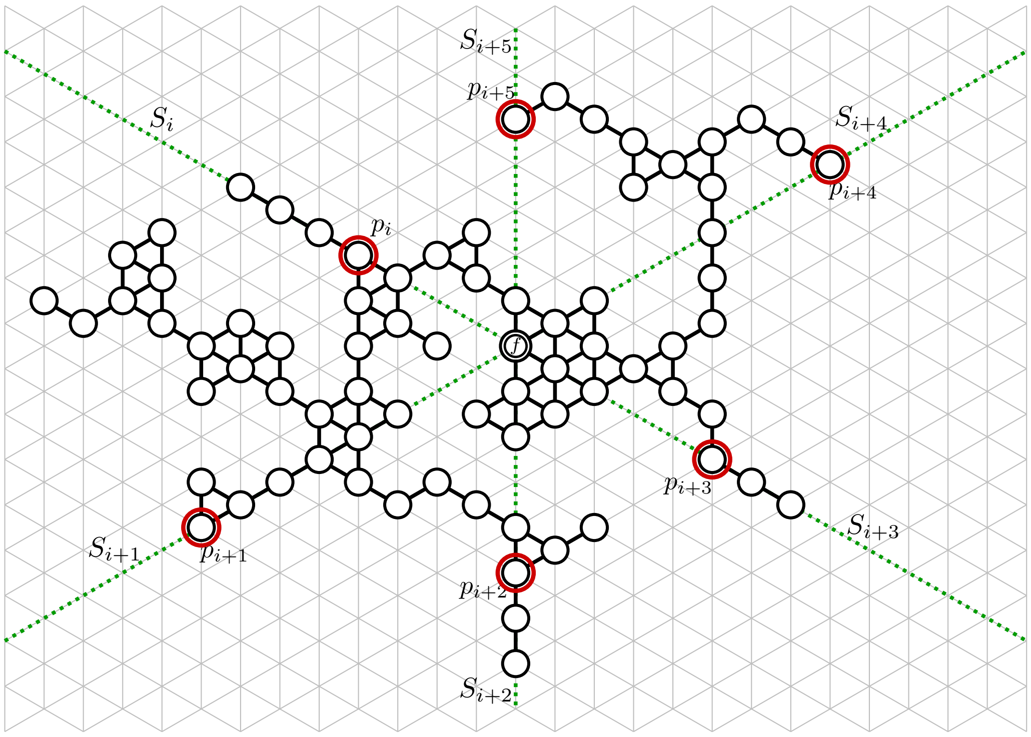


Figure 6 Illustration of the spines extending from the immobile agent f. The six spines $S_i, \ldots, S_{i+5}$ have lengths $4, 8, 5, 5, 8, 5$ respectively. These spines have respective anchor agents $p_i, \ldots, p_{i+5}$ (in red). As an illustration of the coordinate system, these six anchor agents are at coordinates $(4, 0)$, $(8, 8)$, $(0, 5)$, $(-5, 0)$, $(-8, -8)$ and $(0, -5)$ respectively.

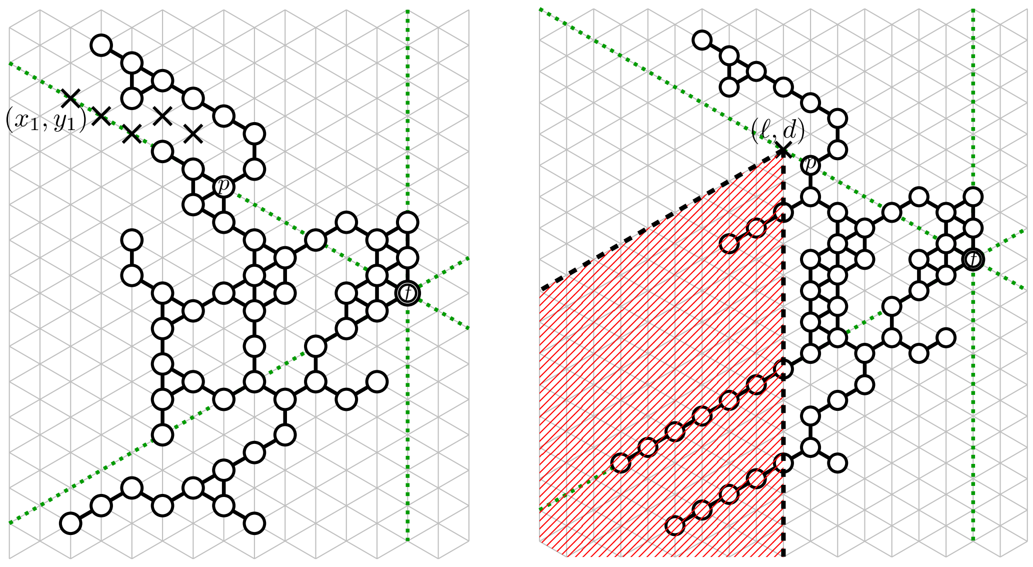
As shown in Figure 6, we notate the six spines using one of the spines as a reference spine. If the reference spine is denoted S_i, where i is an integer modulo 6, then $S_{i+1}, S_{i+2}, \ldots, S_{i+5}$ denote the subsequent spines in a counterclockwise order from S_i.

The proof centers around a specific transformation we call a "comb" operation. This comb operation is applied from one spine (which we refer to as the source spine) to an adjacent spine (which we refer to as the target spine), and has the effect of "pushing" the agents between the two spines towards the target spine.

Our system exhibits reflection symmetry and 6-fold rotational symmetry, so this comb operation can be defined in $6 \times 2 = 12$ different ways. However, for simplicity of discussion, we will only define the comb operation in one orientation, specifically on the left side, downwards. This is a comb from the spine going in the up-left direction to the spine going into the down-left direction. We rotate or reflect the configuration freely, depending on which pair of adjacent spines we want to comb between.

6.1 The comb operation

We define our two-dimensional coordinate system $(lane, depth)$ relative to the source spine, assumed to be going in the up-left direction. A position $(\ell, 0)$ for $\ell \geq 0$ refers to the position on the source spine ℓ steps away from the immobile agent. If $\ell < 0$, this refers to the position $-\ell$ steps in the direction of the spine directly opposite the source spine. A position (ℓ, d) refers to the location d steps downwards from position $(\ell, 0)$. Thus, d denotes the (signed) distance of the position from the source spine.



(a) Illustration of a spine comb (Definition 40), which in this case is a comb over the sequence (Definition 34) denoted by crosses in the Figure.

(b) After combing (ℓ, d), position (ℓ, d) is combed (Definition 29). The shaded region is the residual region of (ℓ, d) (Definition 28)

Figure 7 The comb operation is applied in to the five points marked with crosses in Figure 7a from left to right in sequence, starting from (x_1, y_1). Figure 7b illustrates the end result.

Before we define the comb operation, the following definitions tells us what can and cannot be combed.

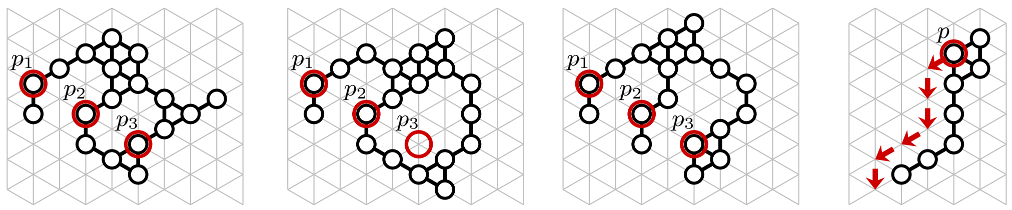
▶ **Definition 28** (Residual Region). *Consider a position (ℓ, d) and the diagonal half-line extending down-left from (ℓ, d), including (ℓ, d) itself. The* residual region *of this position refers to the set of all positions on or below this half-line (Figure 7b).*

▶ **Definition 29** (Combed). *For $\ell > 0$ and $d \geq 0$, we say a position (ℓ, d) is* combed *(Figure 7b) if:*

1. *All sites directly above a topmost agent of the residual region of (ℓ, d) are empty.*
2. *All agents in the residual region of (ℓ, d) form straight lines stretching down and left.*
3. *Consider the column of sites directly to the right of the residual region of (ℓ, d). Each of the abovementioned lines of agents stretches down and left from an agent from this column with no agent directly below.*

▶ **Definition 30** (Combable). *For $\ell > 0$ and $d \geq 0$, we say a position (ℓ, d) is* combable *if:*

1. *The position $(\ell + 1, d + 1)$, which is one step diagonally down-left from (ℓ, d), is combed.*
2. *The site directly above (ℓ, d) is empty.*



(a) Shift p_1, p_2 down-right. **(b)** Shift p_1, p_2 down-right. **(c)** Shift p_2, p_3 up-left. **(d)** p is non-shiftable.

Figure 8 Figures 8a, 8b and 8c illustrate different cases for the "shift" operation. In all of these images, sites p_1 and p_2 are shiftable agents, while p_3 is not.

With this, we can define the comb procedure. Aside from operating only below a given depth, this comb procedure is identical to the process used the proof of irreducibility in [7]. As this operation is covered in detail in said paper, we will be brief with its explanation here.

The comb operation: The comb procedure (applied to a combable position (ℓ, d)) has two phases, line formation and line merging. After the line formation phase, the first two conditions for (ℓ, d) to be combed will be satisfied by the configuration (Figure 9a). The line merging phase gives us the third condition (Figure 9b).

Line formation Let L denote the set of agents on lane ℓ on or below (ℓ, d). The agents in L can be grouped into connected components within L. The line formation phase operates from top to bottom on L, removing the topmost agent of each component with size greater than 1 at each turn, until every component on L has size 1. We maintain the invariant that $(\ell + 1, d + 1)$ is combed after each turn, while reducing the number of agents in L by 1.

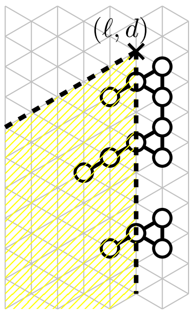
We call a site "shiftable" if there is an agent on the site and it has exactly two neighboring agents, one directly below and one directly up-right of it. In a turn, there are two possible cases. Denote by p the topmost agent of the topmost component with size greater than 1.

If p is shiftable, we apply what we call a "shift", which moves a set of agents on a line either down-right or up-right, so that p either becomes unoccupied or non-shiftable. To apply a shift, we consider the sequence of sites $p = p_1, p_2, \ldots$, where each site p_{i+1} is exactly two steps down-right of site p_i. Let k be the largest integer such that all of the sites $p_1, p_2, \ldots, p_k$ are shiftable. Figures 8a, 8b and 8c illustrate examples where $k = 2$. Consider the first non-shiftable site p_{k+1} in the sequence, and the sites directly above and directly down-left of p_{k+1}, which we will call p^U_{k+1} and p^{DL}_{k+1} respectively. If p_{k+1} is unoccupied or either of p^U_{k+1} or p^{DL}_{k+1} are occupied, then moving p_k one step down-right is a valid move (Figures 8a, 8b). We can thus go backwards through the sequence from p_k to p_1, moving each agent one step down-right, ending with shifting $p = p_1$ one step down-right, so that the site p originally was occupying now becomes unoccupied. On the other hand, if p_{k+1} is occupied but p^U_{k+1} and p^{DL}_{k+1} aren't, as p_{k+1} is non-shiftable, the remaining three neighbors (down, down-right and up-right) must form a single component, meaning moving p_{k+1} one step up-left is a valid move (Figure 8c). We can then subsequently move each agent from p_{k+1} to p_2 one step up-left, culminating in $p = p_1$ becoming non-shiftable, leading in to the second case which we will describe next. Note that after a shift, the invariant that $(\ell + 1, d + 1)$ is combed still holds.

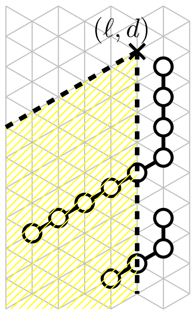
If p is not shiftable, by the invariant we maintain, the sites above, up-left and down-left of p must be unoccupied, while the site directly below p is occupied. Thus, if the site up-right of p is occupied, so must the site down-right of p. As $(\ell + 1, d + 1)$ is combed, the component on L p belongs to will have no agent down-left of it, except for potentially one line of agents extending from the bottommost agent of the component. The agent at p can thus be moved down-left, down along the component on L p belongs to, then down-left to reach the end of the beforementioned line if it exists, and down once more to join the end of this line (Figure 8d). In all, this reduces the number of agents in L by 1, while maintaining the invariant that $(\ell+1, d+1)$ is combed.

Thus, after the line formation phase, every component in L will have exactly 1 agent, while $(\ell+1, d+1)$ remains combed. As the site above (ℓ, d) is empty, the first two conditions for (ℓ, d) being combed are satisfied. The line merging phase will give us the third condition. Figures 9a and 9b illustrate configurations before and after the line merging phase.

Line merging Let C denote the column of sites directly to the right of (ℓ, d). The lines extending down and left in the residual region of (ℓ, d) may extend from agents in C that are not the bottommost agents of their respective components. To fix this, the line merging phase processs these lines from the lowest to the highest. To move a line downwards by one step, the agents of the line are shifted down one by one, starting from the rightmost agent of the line and ending with the agent on the end of the line. These moves are always possible as long as there is no line directly below the current line. If there is a line directly below, we merge the current line into the line below by moving the agents one at the time to the end of the line below with a straightforward sequence of moves, starting with the leftmost agent of the current line.



(a) After line formation, before line merging.



(b) After line merging.

Figure 9 The results after the line formation and line merging phases when the comb procedure is applied to a combable position (ℓ, d).

The description of the comb procedure above gives us the following Lemma:

▶ **Lemma 31.** *After executing a comb operation on a combable position (ℓ, d), the position (ℓ, d) will be combed (Definition 29).*

The following Lemma states that combs only affect sites below it.

▶ **Lemma 32** (Unaffected Region Above). *Consider the two half-lines extending down-left and down-right from a combable position (ℓ, d) as in Figure 10. A comb operation on (ℓ, d) will not affect (will not move any agent into or out of) any site above these lines, not including the lines themselves. In addition, if the site $(\ell-1, d)$ (one step directly down-right) is occupied, no site on the half-line going down-right from $(\ell-1, d)$, including $(\ell-1, d)$ itself, will be affected either.*

Proof. In the comb procedure, aside from the "shift" moves, all of the moves occur only within the residual region of (ℓ, d). The shift moves only go down-right or up-left, and if

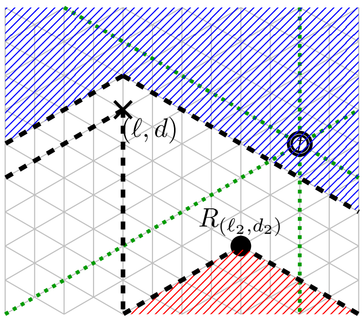


Figure 10 Illustration of the unaffected region above the position to be combed (ℓ, d) from Lemma 32, and an unenterable region $R_{(\ell_2,d_2)}$ from Lemma 38 corresponding to some position (ℓ_2, d_2) strictly to the right of (ℓ, d). Both of these regions include the boundaries drawn in the Figure.

the shift originates from some position p, the shift does not move any agent up-left from p. Hence, the shifts also do not affect any site above the two half-lines described in the Lemma.

We observe that the only part of the procedure that can affect agents on the half-line going down-right from $(\ell - 1, d)$ is a potential shift move on an agent on position (ℓ, d). However, if $(\ell - 1, d)$ is occupied, (ℓ, d) will not be shiftable, and so this shift move will not occur. ◀

To apply a sequence of comb operations to "push" agents down towards the target spine, we only need to find a (not necessarily straight) "line" of vacant positions. The following definition and lemma makes this more formal.

▶ **Lemma 33** (Combable Sequence). *Let $((x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k))$ be a sequence where each item in the sequence represents a $(lane, depth)$ pair. We call this a* combable sequence *if:*

1. *x_1 vertically coincides with the leftmost agent of the configuration.*
2. *$x_{i+1} = x_i - 1$ for all $i \in \{1, 2, \ldots, k - 1\}$ and $x_k > 0$.*
3. *$y_i \geq 0$ for all $i \in \{1, 2, \ldots, k\}$.*
4. *$y_{i+1} \in \{y_i, y_i - 1\}$ for all $i \in \{2, \ldots, k\}$.*
5. *The locations $(x_i, y_i - 1)$ are all vacant.*
6. *For each $i \in \{1, 2, \ldots, k\}$, if $y_i = 0$, then the position $(x_i - 1, 0)$ must be occupied by an agent.*

An example of such a sequence is illustrated in Figure 7a.

▶ **Definition 34** (Combing a Sequence). *If a sequence $((x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k))$ is combable, combing it refers to combing each pair (x_i, y_i) in succession. The following lemma justifies that this is always possible.*

▶ **Lemma 35** (Combability of Each Step in a Sequence). *When combing a combable sequence $((x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k))$ as described in Definition 34, when (x_i, y_i) is the next position to be combed, (x_i, y_i) will be combable.*

Proof. We first note that for any i, by the definition of a combable sequence, the location directly above (x_i, y_i) must be empty, and if $y_i = 0$, then $(x_i - 1, 0)$ is occupied. These two conditions continue to be true even as combs $1, \ldots, i - 1$ are executed, as by Lemma 32, none of these prior combs will affect $(x_i, y_i - 1)$ or $(x_i - 1, y_i)$. This covers two of the conditions necessary for (x_i, y_i) to be combable.

When $i = 1$, (x_1, y_1) is clearly combable as there are no agents to the left of x_1, and the site directly above (x_1, y_1) is empty. For $i \geq 2$, as (x_{i-1}, y_{i-1}) was combed in the previous step, (x_{i-1}, y_{i-1}) is combed (Lemma 31). There are two cases for y_i. If $y_i = y_{i-1} - 1$, then $(x_i + 1, y_i + 1) = (x_{i-1}, y_{i-1})$ is combed. On the other hand, if $y_i = y_{i-1}$, then $(x_i + 1, y_i) = (x_{i-1}, y_{i-1})$ is combed. As $(x_i, y_i - 1)$ is unoccupied, every location starting from $(x_i + 1, y_i)$ extending down-left must also be unoccupied, which implies $(x_i + 1, y_i + 1)$ is combed. This gives us our final condition, so (x_i, y_i) is combable. ◀

In addition to the two properties of the comb operation given as Lemmas 31 and 32, we state and show a few more properties of the comb operation that we will use later in the proof.

▶ **Lemma 36** (Combing and Spine Lengths). *After executing a comb on some position (ℓ, d) where $d < \ell$, the length of the spine going down-left will be at most $\ell - 1$.*

Proof. Let S denote the spine going down-left. The coordinates (ℓ, ℓ) denotes the position on the spine S vertically below (ℓ, d). If (ℓ, ℓ) is empty after the comb, then every site down-left of (ℓ, ℓ) is also empty, so the spine S has length at most $\ell - 1$. If (ℓ, ℓ) is not empty after the comb, (ℓ, d) being combed ensures that (ℓ, ℓ) and every agent on the spine S down-left of (ℓ, ℓ) are tail agents, so spine S has length at most $\ell - 1$. ◀

▶ **Lemma 37** (Preservation of the Rightmost extent). *Let ℓ denote the lane (x-coordinate) of the rightmost agent of a configuration. After a comb operation is applied, the lanes to the right of ℓ (sites with lane less than ℓ) will continue to be empty.*

Proof. Consider a comb applied to some position (ℓ^*, d). As we enforce that $\ell^* > 0$ for a comb, this position is necessarily to the left of the immobile agent, while the rightmost agent of the configuration must be either on the same lane as the immobile agent or further right. All moves aside from the "shift" moves in a comb procedure of a position (ℓ^*, d) operate only within the residual region of (ℓ^*, d), and so will not affect any site on lane ℓ or further right.

In the shift moves, an agent is only moved rightward (down-right) if it is shiftable. A shiftable agent must have an agent directly up-right of it, so a shift move cannot move an agent rightward of ℓ. ◀

▶ **Lemma 38** (Unenterable Region Below). *Consider a position (ℓ, d) and the diagonal half-lines extending down-left and down-right from (ℓ, d). Consider the region $R_{\ell,d}$ containing every location on or below these lines (Figure 10).*

If the region $R_{\ell,d}$ is unoccupied, if a comb operation is applied on a lane strictly to the left of lane ℓ, $R_{\ell,d}$ will continue to be unoccupied after the comb.

Proof. For the shift movements in the line formation phase, an agent is only moved down-right if it is shiftable, which means it must have an agent directly below it. Thus, $R_{\ell,d}$ cannot be entered from the left side (left of lane ℓ) by this movement unless there is already an agent in $R_{\ell,d}$. In addition, as the shift movements only move agents down-right or up-left, it cannot move agents into $R_{\ell,d}$ from the right side (right of lane ℓ). For the other movements in the comb operation, we only need to consider possibly entry into $R_{\ell,d}$ from the left side, as these movements occur only within the residual region of the comb, which is strictly to the left of (ℓ, d).

In the line formation phase, only down-left and downward movements are used. Down-left movements cannot enter $R_{\ell,d}$, and downward movements only occur when there is an agent directly down-right of the agent to be moved.

In the line merging phase, we simply need to consider the end state of the comb. The end state consists of straight lines stretching down-left from the lane (column) ℓ' one lane right of the lane to be combed. All of the agents in this lane are above $R_{\ell,d}$ after the line formation phase, and as $\ell' \leq \ell$, all lines stretching down-left from these agents will also be above $R_{\ell,d}$. ◀

6.2 Using combs to show ergodicity

Our objective is to show that from any (connected) configuration, there exists a sequence of valid moves to transform the configuration into a straight line with the immobile agent at one end. As valid moves cannot introduce holes into a configuration, and all valid moves between hole-free configurations are reversible, this would imply that one can transform any connected configuration of agents into any hole-free configuration of agents using only valid moves. We proceed by showing that we can always reduce the minimum spine length of any configuration with a series of moves to reach a straight line of agents.

▶ **Lemma 39.** *If the minimum spine length of a configuration is at least* 1*, there exists a sequence of moves to reduce the minimum spine length of the configuration.*

To reduce the minimum spine length of the configuration, we execute spine combs in a specific order. Pick a spine of minimum length and denote it as S_0. We apply spine combs in a counterclockwise order, from S_0 to S_1, followed by S_1 to S_2, and so on. When applying comb a comb operation from a source spine S_i to a target spine S_{i+1}, as always, for ease of analysis, we will treat S_i and S_{i+1} as the spines going in the up-left and down-left directions from the immobile agent respectively.

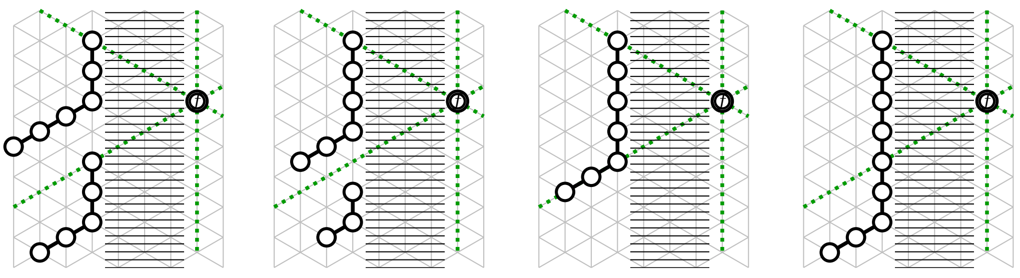
▶ **Definition 40** (Spine Comb). *Let r denote the length of the source spine S_0. Let r_t denote the distance of the furthest out agent on the source spine from the immobile agent (hence the agents of distances $r+1, \ldots, r_t$ are the tail agents).*

A spine comb applies a comb on the combable sequence $((x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k))$, where x_1 vertically coincides with the leftmost agent of the configuration, $x_i = x_1 - i + 1$ for each $i \in \{2, 3, \ldots, k\}$, $x_k = r + 1$, $y_i = 1$ whenever $x_i > r_t$, and $y_i = 0$ when $r + 1 \leq x_i \leq r_t$. From the definition of tail agents one can easily verify that this $(x_i, y_i - 1)$ is vacant for each $i \in \{1, 2, \ldots, k\}$. Figure 7 illustrates configurations before and after a spine comb is applied.

▶ **Lemma 41.** *Consider a spine comb from a source spine of length r. After the spine comb, the position $(r + 1, 1)$ will be combed. Also, the region between (and including) the two half-lines extending up-left and down-left indefinitely from the position $(r + 1, 0)$ will be empty.*

Proof. The last comb operation is on position $(r + 1, 1)$ or $(r + 1, 0)$. If it is the former, $(r + 1, 1)$ will be combed (Lemma 31). If it is on the latter, as none of the comb operations will affect the site $(r, 0)$ directly down-right of the last comb position, $(r, 0)$ will remain occupied by an agent. As $(r + 1, 0)$ is combed, there will be no agents on the diagonal stretching down-left from $(r + 1, 0)$. Hence, $(r + 1, 1)$ is also combed.

The region described in the lemma can be divided into "files", diagonal lines going down-left. Consider any site (x, y) in this region. If (x, y) is in the lowest file of the region (on the diagonal extending down-left from $(r + 1, 0)$), as $(r + 1, 1)$ is combed, (x, y) must



(a) Gap between spines. **(b)** Gap on target spine. **(c)** No gap. **(d)** No gap.

Figure 11 Illustration of the line between the spines going up-left and down-left from the immobile agent. Figures 11a and 11b have gaps in the line (Definition 42), while Figures 11c and 11d do not. Observe that in the cases with no gap, the length of the target spine matches that of the source spine.

be unoccupied. Otherwise, if $(x, y+1)$ is in the same file as some position in the combable sequence, let (x_i, y_i) be the last position in the sequence in the same file as $(x, y+1)$. After (x_i, y_i) is combed, (x, y) must be empty. Subsequent combs will not affect (x, y) by Lemma 32. If $(x, y+1)$ is not in the same file as any position in the combable sequence, (x, y) must be empty as (x_1, y_1) is vertically aligned with the leftmost agent of the configuration. Similarly by Lemma 32, none of the combs will move an agent into (x, y). Hence, (x, y) will be empty after the spine comb in all cases. ◀

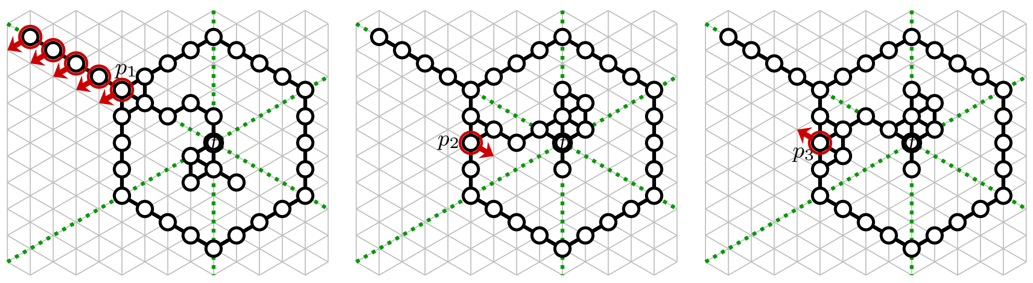
After a spine comb is applied, there are two possible cases, having a gap in the line (defined next), and not having a gap in the line. If there is a gap in the line, we show that we can directly reduce the minimum spine length of the configuration from here, giving us the result of Lemma 39. Hence, we can proceed with the rest of the proof of Lemma 39 assuming that there will never be a gap in the line.

▶ **Definition 42** (Gap in the line). *Let r be the length of the source spine S_i. Consider the vertical line segment of sites from the location of the anchor agent of S_i down to the site on the target spine S_{i+1} of distance r from the center, including the two spine location endpoints. If there is a site on this line segment that is unoccupied by agents, we say that there is a* gap *in the line from the source spine to the target spine.*

▶ **Lemma 43** (Reducing minimum spine length using a gap). *After a spine comb is applied from a source spine of minimum length, if there is a gap in the line from the source spine to the target spine, there exists a sequence of moves to reduce the minimum spine length of the configuration.*

Proof. Let r denote the length of the source spine S_i. Suppose that there is an unoccupied site (r, d) on this line segment. If the unoccupied site on the line segment is on the source spine S_i (which actually never happens), as every site on the spine of distance greater than r will be unoccupied by Lemma 41, the new minimum spine length would be at most $r-1$. If the unoccupied site is on the target spine S_{i+1}, as $(r+1, 1)$ is combed, every site of the target spine of distance greater than this unoccupied site would also be unoccupied. Hence the minimum spine length would also have decreased to at most $r-1$ in this case.

If the unoccupied site (r, d) lies strictly between the source spine and the target spine, we apply one more comb on position $(r, d+1)$. Position $(r, d+1)$ is combable as (r, d) is empty, and $(r+1, 1)$ being combed ensures that all sites on the half-line extending down-left



(a) Moving a corner outward. **(b)** Moving a side agent inward. **(c)** Moving a side agent outward.

Figure 12 Possible cases for reducing the minimum spine length from a hexagon with a tail.

from (r, d) are also empty, so $(r+1, d+1)$ is also combed. By Lemma 36, combing $(r, d+1)$ results in the target spine having length at most $r-1$. ◀

Thus, from now on we may assume that whenever a spine comb is executed from a source spine of minimum length, there will be no gaps in the line between the source spine and the target spine. In addition, we assume that the comb operations do not cause any other spine (in particular the spine going downwards) to end up with a spine length below the current minimum spine length, as in this case we have already achieved the result of Lemma 39. The following Lemma shows that such a spine comb "pushes" all of the agents of distance greater than the minimum spine length towards the target spine ore beyond.

▶ **Lemma 44** (Resulting configuration assuming no gap exists). *Suppose that a spine comb is executed from a source spine S_i (of minimal length r) to a target spine S_{i+1}, and assume that there are no gaps in the line between S_i and S_{i+1}. In the resulting configuration, there will be no agent of distance greater than r strictly between the source spine and target spine, or on the source spine itself. Furthermore, the lengths of both the source spine and target spine will now be exactly r.*

Proof. After the spine comb, position $(r+1, 1)$ will be combed, and the region between the down-left and up-left diagonals extending from $(r+1, 0)$ as described in Lemma 41 will be empty. As there is no gap in the line from S_i to S_{i+1}, the only lines extending left and down in the residual region of $(r+1, 1)$ will be on the target spine or below, giving us the first part of this Lemma.

The length of the source spine is r as position $(r, 0)$ is occupied while no position on the source spine beyond that is. For the length of the target spine, the position on the target spine of distance r from the immobile agent is occupied and is not a tail agent, and by Lemma 39, $(r+1, 1)$ being combed implies that the target spine has length at most r. ◀

As a spine comb sets the length of the target spine S_{i+1} to be the same as that of the source spine S_i, which has minimum length, we can continue executing spine combs in a counterclockwise fashion, from S_{i+1} to S_{i+2}, followed by S_{i+2} to S_{i+3}, and so on. We show that after seven of these spine combs which do not create gaps, we will reach a type of configuration we will call a *hexagon with a tail.* Figure 12 illustrates examples of these "hexagon with a tail" configurations, though one should note that it is not necessary for all sites on the outer hexagon to be filled.

▶ **Definition 45** (Hexagon with a Tail). *We say a configuration forms a has the "hexagon with a tail" arrangement of radius r if:*

- *All spines have length exactly r;*
- *There are tail agents on at most one of the spines;*
- *Aside from these tail agents, there are no agents of distance greater than r from the center.*

If $r = 0$, this "regular hexagon" comprises of only the immobile agent. In other words, a hexagon with a tail of radius r has all of the agents extending in a straight line from the immobile agent.

▶ **Lemma 46** (Reaching a Hexagon with a Tail). *After seven spine combs in a counterclockwise order starting from a spine of minimum length r, assuming that no gaps in the lines are formed and that no spine ends up with length below r in the process, we will end up with a hexagon with a tail arrangement of radius r.*

Proof. We will denote the starting spine as S_0, and name the remaining spines S_1 to S_5 in counterclockwise order. The spine combs hence go from S_0 to S_1, from S_1 to S_2 and so on, with the final (seventh) comb being from S_0 to S_1. Spine S_0 is assumed to be a minimum length spine, of length r.

By Lemma 32, a spine comb from spines S_i to S_{i+1} will only affect agents on spines S_i, S_{i+1}, S_{i+2}, and the agents between spines S_i and S_{i+1}, between spines S_{i+1} and S_{i+2}, and between spines S_{i+2} and S_{i+3}. Note that this does include the agents on spine S_{i+3}. Hence, the first four spine combs will not affect the result of the first spine comb from S_0 to S_1.

On the fifth spine comb from S_4 to S_5, as usual without loss of generality we take S_4 to be the spine going up-left and S_5 to be the spine going down-left. Spine S_0 will thus be going downwards and spine S_1 will be going down-right. Due to the effects of the first three combs, there will be no agent further right than the anchor agent of spine S_1. By Lemma 37, while the fifth spine comb may move agents onto spine S_0 or the region between spines S_0 and S_1, none of these agents in the resulting configuration will be further right than the anchor agent of spineS_1.

On the sixth spine comb from S_5 to S_0, taking S_5 to be going up-left and S_0 to be going down-left, consider the position $(-r-1, 0)$, which is one agent down-right of the anchor agent of the down-right spine S_2. The region $R_{-r-1,0}$, as defined in Lemma 38, will be empty after the fifth spine comb, due to what we have just shown to happen after the fifth spine comb. By Lemma 38, this region will continue to be empty after the sixth spine comb.

On the seventh and final spine comb from S_0 to S_1, take S_0 to be going up-left and S_1 to be going down-left. Consider the positions $(r, 0)$ and (r, r), which are on the source spine S_0 and target spine S_1 respectively, of distance r from the center. As a result of the sixth spine comb with Lemma 44, all agents of distance greater than r from the center must lie between (inclusive) the two diagonal lines going up-left from the positions $(r, 0)$ and (r, r). Now, from the position $(r, r+1)$ which lies directly below (r, r) and the position $(-r-1, 0)$, which lies on spine S_3 of distance $r+1$ from the center, we consider the two regions $R_{r,r+1}$ and $R_{-r-1,0}$ as in Lemma 38. Both of these regions are initially empty, and so will remain empty after the seventh comb. By Lemmas 32 and 44, the only place where agents of distance greater than r can be are on the target spine S_1.

As we had assumed that no spine will have ended up with length less than r in the process, this means we have reached a hexagon with a tail arrangement of radius r. ◀

The following Lemma then concludes the proof that we can always reduce the minimum spine length, provided that the current minimum spine length is at least 1.

▶ **Lemma 47.** *From a hexagon with a tail arrangement of radius $r \geq 1$, there exists a sequence of moves to reduce the minimum spine length by 1.*

Proof. Consider the set H of positions of distance exactly r from the center. This set of positions is in the shape of a hexagon. If one of these sites is unfilled, without loss of generality assume this site (r, d) is on the left side of the hexagon (it is not on a corner as all spines have length r). The site $(r, d + 1)$ is combable, which by Lemma 36 gives us a way to reduce the length of the spine going down-left to at most $r - 1$.

If no such gap in H currently exists, we show that we can create such a gap. If $r = 1$, pick any agent on the hexagon H aside from the one on the spine the tail is on. This agent can be moved to a vacant spot between two spines, reducing the minimum spine length to 0.

Otherwise, as the configuration is assumed to be connected, there is a path of agents from the center (immobile) agent to an agent on H. This implies that there is an agent of v_{-2} distance $r - 2$ from the center adjacent to an agent v_{-1} of distance $r - 1$ from the center. Note that if $r = 2$, v_{-2} will be the immobile agent. If v_{-1} is adjacent to a corner agent of the hexagon H, assuming without loss of generality that this corner is on the spine going up-left, we can move this corner agent one step down-left, and if there are any tail agents attached to this corner agent, they can then subsequently be moved one-by-one one step down-left as well (Figure 12a). This reduces the minimum spine length to at most $r - 1$.

If v_{-1} is not adjacent to a corner agent of H, we note that v_{-1} and v_{-2} will share a neighboring site u_{-1} of distance $r-1$ from the center. The sites u_{-1} and v_{-1} share a neighbor agent v_0 on H. If site u_{-1} is unoccupied, agent v_0 can be moved into site u_{-1}, creating a gap in the hexagon H (Figure 12b). If u_{-1} is occupied, v_0 can be moved in the opposite direction of u_{-1}, to a position u_{+1} of distance $r + 1$ from the center, creating a gap in the hexagon H (Figure 12c).

In both of these cases, by reflection and rotational symmetry, without loss of generality, this newly created gap v_0 is on the left wall of the hexagon H, and if the agent was moved to u_{+1}, u_{+1} is directly up-left of v_0. The site directly below v_0 is thus combable, and by Lemma 36, combing this reduces the minimum spine length to at most $r - 1$. ◀

Finally, we show that we can reach a straight line of agents, thus showing ergodicity of the chain since the chain is reversible.

▶ **Lemma 48.** *From any connected configuration of agents with one single immobile agent, there exists a sequence of valid moves to transform this configuration into a straight line of agents with the immobile agent at one end.*

Proof. Applying Lemma 39 repeatedly allows us to arrive at a configuration with minimum spine length 0. Applying Lemma 46 from here gives us a hexagon with a tail arrangement of radius 0, which is a straight line of agents with the immobile agent at one end. ◀

We observe that the direction in which the final tail faces is irrelevant, as there is a simple sequence of moves to change the direction of the tail, by moving the agents one by one to the location of the new tail, starting from the agent at the very end of the initial tail. This thus allows us to conclude Lemma 23, which also implies that the Markov chain is irreducible.

7 Conclusion

In this paper we show how a group of computationally limited agents can autonomously respond to environmental cues in the form of appearing and disappearing stimuli to iteratively perform the desired collective response. The self-induced collective phase changes are performed through local communication and state changes that ensure the collective recovers from multiple, possibly conflicting, signals, overcoming challenges that arise as the agents move, constantly changing the underlying connectivity network through which agents can communicate. We apply this framework to the foraging problem, whereby the appearance of a stimulus indicates the presence of food, triggering the agents to gather and feed, and the stimulus disappearing indicates the depletion of food, triggering a search phase where the agents disperse in search of a new food source. Interestingly the gather and dispersion phases can be implemented with essentially the same algorithm, with a single parameter representing the affinity of agents to be close to other agents, which is known to undergo a genuine phase change (in the physics sense) in both the connected and general settings [7, 30].

This framework should be useful in the context of other stochastic algorithms for programmable matter, such as separation, where heterogeneous agents reorganize into tight-knit homogenous clusters or integrate with other types, depending on the value of a homophily parameter, or alignment, in which agents are oriented and attempt to align with their neighbors depending on an alignment parameter [6, 27]. In each of these cases the collectives are known to undergo phase changes as long as the collectives are sufficiently compressed so that the underlying network is highly connected. It would be interesting to extend our algorithm to such two (or more) parameter systems, where one parameter controls the network connectivity and the other controls the outcome of the task, such as the degree of separation or alignment.

There are many generalizations of our model that would be interesting for further investigation and that would bring us closer to more realistic models of programmable matter and related areas. For example, in this work, the assumption that an agent can change its state and the state of its neighbors during an (atomic) action is justified since we assume a sequential scheduler (e.g., such an assumption also appears in [13, 18]). However, in the presence of a stronger adversarial scheduler, e.g., the asynchronous scheduler [15], one would need a more detailed message passing mechanism to ensure the successful transfer of tokens between agents, and the resulting changes in their states.

The following generalizations of our model would also be of interest in the study of self-induced collective phase changes. Throughout we assume uniform rates for the Poisson clocks associated with agents, but we believe that our results can also accommodate non-uniform constant rates with only minor modifications to the proofs. Last, we assume that agents are all aware of upper bounds on the maximum degree Δ because this allows us to implement our token passing mechanism in an equitable way in which we can upper and lower bound the rates that messages will be spread throughout the evolving network. It is likely that this assumption can be relaxed, although in most settings we had in mind this is a reasonable assumption, arising from the planarity (or low dimensionality) of the models.

Our algorithm currently uses a system of alert tokens to intentionally slow down the rate of growth of a cluster of Aware particles relative to the rate the particles would switch to the Unaware state, so that unaware broadcast waves would always be favored when broadcast waves compete. A question of interest is whether the same can be done without the use of alert tokens, using a simpler scheme of having one type of wave (in this case, the aware wave) propagate a constant factor faster than the other type (unaware waves). Experiments on grid graphs seem to indicate that this can be done, but the proofs are more complex, and it is unclear what types of dynamic graphs such a scheme would be guaranteed to work on (probably something more restrictive than reconfigurable graphs).

References

1 Simon Alberti. Organizing living matter: The role of phase transitions in cell biology and disease. *Biophysical journal*, 14, 2018.

2 Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A stochastic approach to shortcut bridging in programmable matter. In *23rd International Con- ference on DNA Computing and Molecular Programming (DNA)*, pages 122–138, 2017.

3 Chen Avin, Michal Koucký, and Zvi Lotker. Cover time and mixing time of random walks on dynamic graphs. *Random Structures & Algorithms*, 52(4):576–596, 2018.

4 Rodney J. Baxter. *Exactly solved models in statistical mechanics.* Academic Press, 1982.

5 Levent Bayindir. A review of swarm robotics tasks. *Neurocomputing*, 172:292–321, 2016.

6 Sarah Cannon, Joshua J. Daymude, Cem Gökmen, Dana Randall, and Andréa W. Richa. A local stochastic algorithm for separation in heterogeneous self-organizing particle systems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 54:1–54:22, 2019.

7 Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A Markov chain algorithm for compression in self-organizing particle systems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 279–288, 2016.

8 Arnaud Casteigts. Finding structure in dynamic networks, 2018. `arXiv:1807.07801`.

9 Bernard Chazelle. The convergence of bird flocking. *Journal of the ACM (JACM)*, 61(4), 2014.

10 Andrea Clementi, Riccardo Silvestri, and Luca Trevisan. Information spreading in dynamic graphs. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, page 37–46, 2012.

11 Nikolaus Correll and Alcherio Martinoli. Modeling and designing self-organized aggregation in a swarm of miniature robots. *The International Journal of Robotics Research*, 30(5):615–626, 2011.

12 Paolo Dario, Renzo Valleggi, Maria Chiara Carrozza, M. C. Montesi, and Michele Cocco. Microactuators for microrobots: a critical survey. *Journal of Micromechanics and Microengineering*, 2(3):141–157, 1992.

13 Joshua J. Daymude, Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Christian Scheideler, and Andréa W. Richa. Convex hull formation for programmable matter. In *21st International Conference on Distributed Computing and Networking (ICDCN)*, pages 2:1–2:10. ACM, 2020.

14 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The Canonical Amoebot Model: Algorithms and Concurrency Control. In *35th International Symposium on Distributed Computing (DISC)*, volume 209 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, 2021.

15 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The canonical amoebot model: Algorithms and concurrency control. *Distributed Computing*, 36(2):159–192, 2023.

16 Oksana Denysyuk and Luís Rodrigues. Random walks on evolving graphs with recurring topologies. In *Distributed Computing*, pages 333–345, 2014.

17 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: Amoebot - a new model for programmable matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 220–222, 2014.

18 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Universal shape formation for programmable matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016.

19 Michael Dinitz, Jeremy T. Fineman, Seth Gilbert, and Calvin Newport. Smoothed analysis of information spreading in dynamic networks. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC)*, volume 246 of *LIPIcs*, pages 18:1–18:22, 2022.

20 Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, Zhifeng Sun, and Emanuele Viola. On the complexity of information spreading in dynamic networks. In *Proceedings of the 24th Symposium on Discrete Algorithms (SIAM)*, pages 717–736, 2013.

21 Nazim Fatès. Solving the decentralised gathering problem with a reaction–diffusion–chemotaxis scheme. *Swarm Intelligence*, 4(2):91–115, 2010.

22 Nazim Fatès and Nikolaos Vlassopoulos. A robust aggregation method for quasi-blind robots in an active environment. In *ICSI 2011*, 2011.

23 Simon Garnier, Jacques Gautrais, Masoud Asadpour, Christian Jost, and Guy Theraulaz. Self-organized aggregation triggers collective decision making in a group of cockroach-like robots. *Adaptive Behavior*, 17(2):109–133, 2009.

24 Simon Garnier, Christian Jost, Raphaël Jeanson, Jacques Gautrais, Masoud Asadpour, Gilles Caprari, and Guy Theraulaz. Aggregation behaviour as a source of collective decision in a group of cockroach-like-robots. In *Advances in Artificial Life (ECAL)*, pages 169–178, 2005.

25 Bernhard Haeupler and David Karger. Faster information dissemination in dynamic networks via network coding. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, page 381–390, 2011.

26 Walter Hussak and Amitabh Trehan. On termination of a flooding process. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, page 153–155, 2019.

27 Hridesh Kedia, Shunhao Oh, and Dana Randall. A local stochastic algorithm for alignment in self-organizing particle systems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 245, pages 14:1–14:20, 2022.

28 Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, page 513–522, 2010.

29 Fabian Kuhn and Rotem Oshman. Dynamic networks: Models and algorithms. *SIGACT News*, 42(1):82–96, 2011.

30 Shengkai Li, Bahnisikha Dutta, Sarah Cannon, Joshua J. Daymude, Ram Avinery, Enes Aydin, Andréa W. Richa, Daniel I. Goldman, and Dana Randall. Programming active granular matter with mechanically induced phase changes. *Science Advances*, 7, 2021.

31 Jintao Liu, Arthur Prindle, Jacqueline Humphries, Marçal Gabalda-Sagarra, Munehiro Asally, Dong-Yeon D. Lee, San Ly, Jordi Garcia-Ojalvo, and Gürol M. Süel. Metabolic co-dependence gives rise to collective oscillations within biofilms. *Nature*, 523(7562):550–554, 2015.

32 László Lovász. Random walks on graphs. *Combinatorics, P. Erdös is Eighty*, 2(4):1–46, 1993.

33 Anne E. Magurran. The adaptive significance of schooling as an anti-predator defence in fish. *Annales Zoologici Fennici*, 27(2):51–66, 1990.

34 Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21:1087–1092, 1953.

35 Nathan J. Mlot, Craig A. Tovey, and David L. Hu. Fire ants self-assemble into waterproof rafts to survive floods. *Proceedings of the National Academy of Sciences*, 108(19):7669–7673, 2011.

36 Anil Özdemir, Melvin Gauci, Salomé Bonnet, and Roderich Groß. Finding consensus without computation. *IEEE Robotics and Automation Letters*, 3(3):1346–1353, 2018.

37 Arthur Prindle, Jintao Liu, Munehiro Asally, San Ly, Jordi Garcia-Ojalvo, and Gürol M. Süel. Ion channels enable electrical communication in bacterial communities. *Nature*, 527(7576):59–63, 2015.

38 Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics*, pages 10–20, 2005.

39 William Savoie, Sarah Cannon, Joshua J. Daymude, Ross Warkentin, Shengkai Li, Andréa W. Richa, Dana Randall, and Daniel I. Goldman. Phototactic supersmarticles. *Artificial Life and Robotics*, 23(4):459–468, 2018.

40 Thomas C. Schelling. Dynamic models of segregation. *The Journal of Mathematical Sociology*, 1(2):143–186, 1971.

41 Onur Soysal and Erol Şahin. Probabilistic aggregation strategies in swarm robotic systems. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS)*, pages 325–332, 2005.

42 Tommaso Toffoli and Norman Margolus. Programmable matter: Concepts and realization. *Physica D: Nonlinear Phenomena*, 47(1):263–272, 1991.

43 David H. Wolpert. The stochastic thermodynamics of computation. *Journal of Physics A: Mathematical and Theoretical*, 52(19):193001, 2019.

44 Hui Xie, Mengmeng Sun, Xinjian Fan, Zhihua Lin, Weinan Chen, Lei Wang, Lixin Dong, and Qiang He. Reconfigurable magnetic microrobot swarm: Multimode transformation, locomotion, and manipulation. *Science Robotics*, 4(28):eaav8006, 2019.