Bio-Inspired Energy Distribution for Programmable Matter

Joshua J. Daymude
Arizona State University
Computer Science, CIDSE
Tempe, AZ, USA
jdaymude@asu.edu

Andréa W. Richa
Arizona State University
Computer Science, CIDSE
Tempe, AZ, USA
aricha@asu.edu

Jamison W. Weber
Arizona State University
Computer Science, CIDSE
Tempe, AZ, USA
jwweber@asu.edu

ABSTRACT

In systems of *active programmable matter*, individual modules require a constant supply of energy to participate in the system's collective behavior. These systems are often powered by an *external energy source* accessible by at least one module and rely on *module-to-module power transfer* to distribute energy throughout the system. While much effort has gone into addressing challenging aspects of power management in programmable matter hardware, algorithmic theory for programmable matter has largely ignored the impact of energy usage and distribution on algorithm feasibility and efficiency. In this work, we present an algorithm for *energy distribution* in the *amoebot model* that is loosely inspired by the growth behavior of *Bacillus subtilis* bacterial biofilms. These bacteria use chemical signaling to communicate their metabolic states and regulate nutrient consumption throughout the biofilm, ensuring that all bacteria receive the nutrients they need. Our algorithm similarly uses communication to inhibit energy usage when there are starving modules, enabling all modules to receive sufficient energy to meet their demands. As a supporting but independent result, we extend the amoebot model's well-established *spanning forest primitive* so that it *self-stabilizes* in the presence of crash failures. We conclude by showing how this self-stabilizing primitive can be leveraged to compose our energy distribution algorithm with existing amoebot model algorithms, effectively generalizing previous work to also consider energy constraints.

CCS CONCEPTS

• **Theory of computation** → **Self-organization**; *Distributed algorithms*; • **Applied computing** → *Biological networks.*

KEYWORDS

programmable matter, self-organization, distributed algorithms, biologically-inspired algorithms, biofilms, energy

ACM Reference Format:
Joshua J. Daymude, Andréa W. Richa, and Jamison W. Weber. 2021. Bio-Inspired Energy Distribution for Programmable Matter. In *International Conference on Distributed Computing and Networking 2021 (ICDCN '21), January 5–8, 2021, Nara, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3427796.3427835

ICDCN '21, January 5–8, 2021, Nara, Japan

ACM ISBN 978-1-4503-8933-4/21/01…$15.00
https://doi.org/10.1145/3427796.3427835

1 INTRODUCTION

The goal for *programmable matter* [25] is to realize physical materials that can dynamically change their physical properties on command, acting autonomously or based on user input. In *active* systems, the composing modules (or "particles") of programmable matter are often envisioned and designed to be simple, homogeneous units capable of internal computation, inter-module communication, and movement. These modules require a constant supply of energy to function, but as the number of modules per collective increases and individual modules are miniaturized from the centimeter/millimeter-scale [11, 14, 23] to the micro- and nano-scale [9, 16], traditional methods of robotic power supply such as internal battery storage and tethering become infeasible.

Programmable matter systems instead make use of an *external energy source* accessible by at least one module and rely on *module-to-module power transfer* to supply the system with energy [4, 11, 23]. This external energy can be supplied directly to one or more modules in the form of electricity, as in [11], or may be ambiently available as light, heat, sound, or chemical energy in the environment [18, 21]. Since energy may not be uniformly accessible to all modules in the system, a strategy for *energy distribution* — or sharing energy between modules such that all modules eventually obtain the energy they need to function — is imperative but does not come for free. Significant energy loss can occur in module-to-module transfer depending on the method used, and even with perfect transfer successive voltage drops between modules can limit the number of modules that can be powered from a single source [11]. Module geometry may further complicate the problem by introducing short circuits, adding further constraints to power routing algorithms [4].

Algorithmic theory for programmable matter has largely ignored the role of energy (with notable exceptions, such as [9, 23]), focusing primarily on characterizing the minimal capabilities individual modules need to collectively achieve desired system-level self-organizing behaviors. Across models of active programmable matter — including population protocols [1], the nubot model [27], mobile robots [10], hybrid programmable matter [12, 13], and the amoebot model [5, 7] — most works either develop algorithms for a desired behavior and bound their time complexity or prove that a given behavior cannot be achieved within the given constraints. To the extent of our knowledge, papers on these models have only mentioned energy to justify constraints (e.g., why a system should remain connected [19]) and have never directly treated the impact of energy usage and distribution on an algorithm's efficiency. In contrast, both programmable matter practitioners and the modular and swarm robotics literature view energy constraints as influential aspects of algorithm design [2, 15, 20, 22, 26].

In this work, we present an algorithm for energy distribution in the amoebot model that is loosely inspired by the growth behavior of *Bacillus subtilis* bacterial biofilms [17, 24]. We assume that all particles in the system require energy to perform their actions but only some have access to an external energy source. Naive distribution strategies such as fully selfish or fully altruistic behaviors have obvious problems: in the former, particles with access to energy use it all and starve the others, while in the latter no particle ever knows when it is safe to use its stored energy. This necessitates a strategy in which particles shift between selfish and altruistic energy usage depending on the needs of their neighbors. Our algorithm mimics the way bacteria use long-range communication of their metabolic stress to temporarily inhibit the biofilm's energy consumption, allowing for nutrients to reach starving bacteria and effectively solving the energy distribution problem.

1.1 Biological Inspiration

Our strategy of shifting between selfish and altruistic energy usage to achieve energy distribution is loosely inspired by the work of Liu and Prindle et al. [17, 24] on the growth behavior of colonies of *Bacillus subtilis* bacteria, which we summarize here for the sake of completeness. These bacteria form densely packed *biofilm colonies* when they become metabolically stressed (i.e., when they become nutrient scarce and begin to starve). These bacteria consume *glutamine*, which is produced from a combination of substrates *glutamate* and *ammonium*. Glutamate is sourced from the environment outside of the biofilm, whereas ammonium is produced by individual bacterium. However, because ammonium can freely diffuse across a bacterium's cell membrane and be lost to its surroundings, production of ammonium is known as the *futile cycle*. The futile cycle is detrimental for bacteria on the biofilm's periphery, as they lose all their ammonium to the external medium. Once a biofilm colony is formed, however, bacteria in the biofilm's interior are shielded from the futile cycle by those on the periphery. This creates a symbiotic co-dependence: bacteria in the interior are reliant on glutamate passed from the periphery, while bacteria on the periphery are reliant on ammonium produced by the interior.

As the biofilm grows, overall glutamate consumption in the periphery increases, limiting the amount of glutamate that permeates into the interior of the colony. This causes interior bacteria to become metabolically stressed. Thus, in order to regulate glutamate consumption on the periphery, interior bacteria communicate their metabolic states to the peripheral bacteria via a long-range electrochemical process known as *potassium ion-channel-mediated signaling* [24]. This sudden influx of potassium inhibits a bacterium's glutamate intake and ammonium retention, allowing more nutrients to pass into the biofilm's interior. As a result, the biofilm grows at an oscillating rate rather than a constant one, despite the fact that there is plentiful glutamate in the environment. This emergent oscillation enables continuous distribution of nutrients throughout the colony, effectively solving the energy distribution problem.

1.2 The Amoebot Model

In the *amoebot model* [5, 7], programmable matter consists of individual, homogeneous computational elements called *particles*. Any structure that a particle system can form is represented as a subgraph of an infinite, undirected graph $G = (V, E)$ where V represents all relative positions a particle can occupy and E represents all possible adjacencies between particles.[1] Each node can be occupied by at most one particle. The *geometric amoebot model* is a standard model variant that assumes $G = G_\Delta$, the triangular lattice (see Figure 1a).

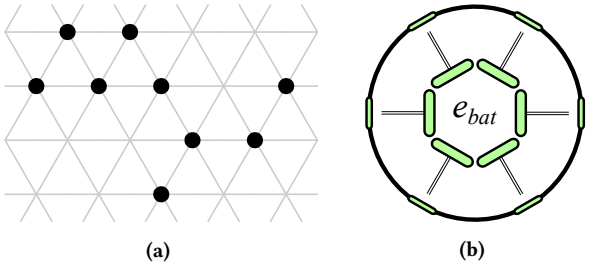


Figure 1: (a) Particles shown as black circles on the triangular lattice G_Δ, shown in gray. (b) A particle's energy anatomy. Energy is transferred between particles at their contact points, shown as green markers on the particle's periphery. A particle's battery e_{bat} stores energy for its own use and for sharing with its neighbors.

Two particles occupying adjacent nodes are said to be *neighbors*. Although each particle is *anonymous*, lacking a unique identifier, a particle can locally identify any given neighbor by its label for the edge between them. Each particle has a constant-size local memory that it and its neighbors can directly read from and write to for communication. However, particles do not have any global information, including a shared coordinate system or orientation.

The system progresses asynchronously through *atomic actions*. In the amoebot model, an atomic action corresponds to a single particle's activation in which it can perform a constant amount of local computation involving information it reads from its local memory and its neighbors' memories and write updates to its neighbors' memories. We assume these actions preserve *atomicity*, *isolation*, and *fairness*. Atomicity requires that an action either completes successfully or is aborted (e.g., due to a conflict) and completely undone. A set of concurrent actions preserves isolation if they do not interfere with each other; i.e., if their concurrent execution produces the same end result as if they were executed in any sequential order. Fairness requires that each particle successfully completes an action infinitely often.

It is well known that if a distributed system's actions are atomic and isolated, any set of such actions can be *serialized* [3]; i.e., there exists a sequential ordering of the successful (non-aborted) actions that produces the same end result as their concurrent execution. Thus, while in reality many particles may be active concurrently, it suffices when analyzing amoebot algorithms to consider the sequential setting where only one particle is active at a time. By our fairness assumption, if a particle P is inactive at time t in the activation sequence, P will be (successfully) activated again at some time $t' > t$. An *asynchronous round* is complete once every particle has been activated at least once.

[1] We omit several core features of the amoebot model such as movements since they are not needed in this work; see [5] for a full description of the model.

Energy Distribution. In addition to the standard model, we introduce terminology specific to the problem of energy distribution. Each particle P has an *energy battery* denoted $P.e_{bat}$ with constant capacity $\kappa > 0$ (see Figure 1b). The battery represents stored energy P can use for performing actions or for sharing with its neighbors. Particles with access to an external energy source can harvest energy into their batteries directly, while those that do not depend on their neighbors to share with them. In either case, each particle can transfer at most a constant $\alpha > 0$ units of energy per activation.

1.3 Our Results

An instance of the *energy distribution problem* has the form $(\mathcal{P}, \kappa, \delta)$ where $\mathcal{P}$ is a finite connected particle system, κ is the capacity of each particle's battery, and energy demand $\delta(P, i)$ denotes the energy cost for a particle P to perform its i-th action. For convenience, we will use $\delta(P)$ to refer to the energy cost for P to perform its next action. An instance is *valid* if (1) $\mathcal{P}$ contains one or more "root" particles with access to external energy sources and all non-root particles are initially "idle" and (2) for all particle actions, $\delta(\cdot, \cdot) \leq \kappa$; i.e., no energy demand exceeds the batteries' energy capacity. A particle P is *stressed* if the energy level of its battery is strictly less than the demand for its next action, i.e., if $P.e_{bat} < \delta(P)$. An action a of a particle P is *enabled* if, barring any energy considerations, P is able to perform action a. A local, distributed algorithm $\mathcal{A}$ *solves* a valid instance of the energy distribution problem in time t if, when each particle executes $\mathcal{A}$ individually, no particle remains stressed for more than t asynchronous rounds and at least one particle performs an enabled action every t asynchronous rounds.

In Section 2, we present Energy-Sharing: a local, distributed algorithm that solves the energy distribution problem in $O(n)$ asynchronous rounds (Theorem 2.8), where n is the number of particles in the system. This algorithm is asymptotically optimal when the number of external energy sources is fixed (Theorem 2.9). We then show simulation results in Section 3, demonstrating that without the biofilm-inspired communication of particles' energy states, Energy-Sharing fails to distribute sufficient energy throughout the system.

In Section 4, we consider the impact of crash faults on the correctness and runtime of our algorithm. Our fault mitigation strategy relies on a new algorithmic primitive called Forest-Prune-Repair that locally repairs the system's underlying communication structure after a particle crashes. This repair primitive is in fact of independent interest, as it extends the amoebot model's well-established *spanning forest primitive* [5] to be self-stabilizing in the presence of crash failures. Finally, we show how Forest-Prune-Repair can be used to compose other amoebot algorithms with our Energy-Sharing algorithm. This effectively generalizes all previous work on the amoebot model to also consider energy constraints.

2 THE ENERGY DISTRIBUTION ALGORITHM

In this section, we present algorithm Energy-Sharing for energy distribution in self-organizing particle systems. At a high level, this algorithm works as follows. After some initial setup, each particle continuously loops through a sequence of three phases: the communication phase, the sharing phase, and the usage phase. In the *communication phase*, particles propagate signals to communicate the energy states of stressed particles, analogous to the long-range electrochemical signaling via potassium ion channels in the biofilms. Particles then attempt to harvest energy from an external energy source or transfer energy to their neighbors in the *sharing phase*. Finally, particles spend their stored energy to perform actions according to their collective behavior in the *usage phase*. Note that the system is not synchronized and each particle progresses through these phases independently.

Section 2.1 details the setup and phases of Energy-Sharing. We then analyze this algorithm's correctness and runtime in Section 2.2. Complete pseudocode was omitted due to space constraints but can be found in the full version of this paper [6].

2.1 Algorithm Energy-Sharing

The Setup Phase. Recall that particle system $\mathcal{P}$ is connected. Particles with access to an external energy source are roots, and the rest are idle. This phase organizes $\mathcal{P}$ as a spanning forest $\mathcal{F}$ of trees rooted at the root particles. These trees facilitate an analogy to the potassium ion signaling that the bacteria use to communicate when they are metabolically stressed (discussed further in the communication phase). To form $\mathcal{F}$, we make use of the well-established *spanning forest primitive* [5]. If a particle P is idle, it checks if it has a root or active neighbor Q. If so, P becomes active and updates its parent pointer to $P.\text{parent} \leftarrow Q$. This repeats until all particles are active, yielding a spanning forest $\mathcal{F}$.

The Communication Phase. The communication phase facilitates the long-range communication of particles' energy states analogous to the biofilm's potassium ion signaling. This is achieved by sending signals along a particle's tree in the spanning forest $\mathcal{F}$ constructed in the setup phase. In particular, any active particle P that is stressed — i.e., $P.e_{bat} < \delta(P)$ — sets a *stress flag* that remains until P is no longer stressed. Any particle that has a child in its tree with their stress flag set also sets their stress flag, effectively propagating this signal up to its tree's root particle. When the root particle receives this stress signal (or if it is itself stressed), it sets an *inhibit flag*, initiating a broadcast to the rest of the tree. Any particle whose parent in the tree has their inhibit flag set also sets their inhibit flag, propagating this inhibition signal throughout the tree. In the usage phase, inhibited particles will be stopped from spending their energy to perform actions, allowing more energy to pass on to the stressed particles. As we will show in the simulations of Section 3, omitting this phase can result in the indefinite starvation of many of the system's particles.

Signal resets behave analogously to how they are set. Any non-root particle that is not stressed — i.e., $P.e_{bat} \geq \delta(P)$ — and has no children with their stress flags set will reset its stress flag. Once a root no longer has any children with stress flags (and it is itself not stressed), it resets its inhibit flag. Any particle whose parent does not have its inhibit flag set resets its own inhibit flag, and so on.

The Sharing Phase. During the sharing phase, particles harvest energy from external energy sources and transfer energy to their neighbors, if possible. A root particle begins the sharing phase by harvesting $\min\{\alpha, \kappa - P.e_{bat}\}$ units of energy from its external energy source. Any particle P — root or active — then checks to see if it has sufficient energy to share (i.e., $P.e_{bat} \geq \alpha$) and if any of its children in the spanning forest $\mathcal{F}$, say Q, need energy (i.e.,

$Q.e_{bat} < \kappa$). If so, P transfers $\min\{\alpha, \kappa - Q.e_{bat}\}$ units of energy to Q in keeping with the assumption from Section 1.2 that each particle can transfer at most α units of energy per activation.

The Usage Phase. In the usage phase, particles spend their energy to perform actions as required by their collective behavior. Suppose that a is the next action a particle P wants to perform; recall that its energy cost is given by $\delta(P)$. If P has sufficient stored energy to perform this action — i.e., $P.e_{bat} \geq \delta(P)$ — and P does not have its inhibit flag set, then P can spend the required energy and perform action a. Otherwise, P forgoes any action in this activation.

2.2 Analysis

We now prove the correctness and bound the runtime of the Energy-Sharing algorithm. We begin with two straightforward results regarding the setup and communication phases. We state the first without proof, as it follows directly from the analysis of the spanning forest primitive [5].

Lemma 2.1. *All idle particles in the system become active and join the spanning forest $\mathcal{F}$ within n asynchronous rounds, where n is the number of particles in the system.*

For the remainder of our analysis we focus on a single tree $\mathcal{T} \in \mathcal{F}$, which suffices since particles in any single tree act independently from all particles in other trees of $\mathcal{F}$.

Lemma 2.2. *Suppose a particle P in tree $\mathcal{T} \in \mathcal{F}$ is stressed; i.e., $P.e_{bat} < \delta(P)$. If tree $\mathcal{T}$ has depth $d_{\mathcal{T}}$, then all particles in $\mathcal{T}$ will have their inhibit flags set within $2d_{\mathcal{T}}$ asynchronous rounds.*[2]

Proof. Since every particle is activated at least once per asynchronous round, P will be activated within one round and will set its stress flag since $P.e_{bat} < \delta(P)$. Recall that stress flags are propagated up to the root by parents setting their stress flags when they see a child with its stress flag set. There can be at most $d_{\mathcal{T}} - 2$ ancestors of P strictly between P and the root. At least one more ancestor will set its stress flag per asynchronous round, so in at most $d_{\mathcal{T}} - 2$ rounds a child of the root will have its stress flag set.

Within one additional round, the root will be activated and will set its inhibit flag. Inhibit flags are then propagated from the root to all its descendants: in each round, any child that sees its parent's inhibit flag set will also set its own inhibit flag. The longest root-to-descendant path in $\mathcal{T}$ is of length $d_{\mathcal{T}}$, so in at most $d_{\mathcal{T}}$ rounds all particles in $\mathcal{T}$ will have their inhibit flags set. □

Lemma 2.2 shows that when a tree contains at least one stressed particle, every particle in the tree eventually becomes inhibited. This inhibition remains until all stressed particles *recharge*, i.e., until they receive the energy they need to perform their next action. The usage phase prohibits any inhibited particle from spending its energy on actions, so it suffices when bounding the recharge time to analyze how energy is shared within the tree.

In particular, we want to bound the worst case time for a stressed particle in a given tree $\mathcal{T}$ to recharge once all particles in $\mathcal{T}$ are inhibited. We make three observations that make this analysis more tractable. First, we assume that all particles in $\mathcal{T}$ begin this recharging process with empty batteries and need to meet maximum energy demand; i.e., we assume $P.e_{bat} = 0$ and $\delta(P) = \kappa$ for all $P \in \mathcal{T}$. Although the particles of $\mathcal{T}$ may have obtained some energy before becoming inhibited, this assumption can only make recharging slower since more energy is needed. Second, we assume $\kappa/\alpha \in \mathbb{N}$, allowing us to assume all energy is transferred in units of size exactly α. This can be easily realized by rounding any given capacity κ up to the next multiple of α, as this can only increase the energy required in recharging. Third, we show in the following lemma that the recharge time in $\mathcal{T}$ is at most the recharge time in a simple path with the same number of particles.

Lemma 2.3. *Suppose $\mathcal{T}$ is a tree of k particles rooted at a particle R with access to external energy. If all k particles are inhibited and initially have no energy in their batteries, then the worst case number of asynchronous rounds to recharge all particles' batteries in $\mathcal{T}$ is at most the worst case number of rounds to do so in a path $\mathcal{L} = (P_1, \ldots, P_k)$ in which P_1 has access to external energy and $P_i.parent = P_{i-1}$ for all $1 < i \leq k$.*

Proof. Given any tree $\mathcal{U}$ of k inhibited particles rooted at a particle R with access to external energy and an activation sequence A of the particles in $\mathcal{U}$, let $t_A(\mathcal{U})$ denote the number of asynchronous rounds required to recharge all particles' batteries in $\mathcal{U}$ with respect to activation sequence A. We use $t(\mathcal{U}) = \max_A\{t_A(\mathcal{U})\}$ to denote the worst case recharge time for $\mathcal{U}$. With this notation, our goal is to show that $t(\mathcal{T}) \leq t(\mathcal{L})$.

Consider the maximal "non-branching" path $(R = P_1, \ldots, P_\ell = P)$ in tree $\mathcal{T}$ starting at the root R such that P_{i+1} is the only child particle of P_i in $\mathcal{T}$ for all $1 \leq i < \ell$. We argue by (reverse) induction on ℓ, the total number of particles in the maximal non-branching path of $\mathcal{T}$. If $\ell = k$, then $\mathcal{T}$ is already a path $\mathcal{L}$ of k particles and we have $t(\mathcal{T}) = t(\mathcal{L})$ trivially. So suppose that $\ell < k$ and for all possible trees $\mathcal{U}$ composed of the same k particles as $\mathcal{T}$ that are rooted at R and have at least $\ell + 1$ particles in their maximal non-branching paths starting at R, we have $t(\mathcal{U}) \leq t(\mathcal{L})$. Our goal is to modify $\mathcal{T}$ to form another tree $\mathcal{T}'$ that is composed of the same particles, is rooted at R, and has exactly one more particle in its maximal non-branching path such that $t(\mathcal{T}) \leq t(\mathcal{T}')$. Since $\mathcal{T}'$ has exactly $\ell + 1$ particles in its maximal non-branching path, the induction hypothesis lets us conclude that $t(\mathcal{T}) \leq t(\mathcal{T}') \leq t(\mathcal{L})$.

With maximal non-branching path $(R = P_1, \ldots, P_\ell = P)$ of $\mathcal{T}$, $P = P_\ell$ is the "closest" particle to R with multiple children, say $Q_1, \ldots, Q_c$ for $c \geq 2$; note that such a particle P must exist since $\ell < k$. Form the tree $\mathcal{T}'$ by reassigning $Q_i.$parent from P to Q_1 for each $2 \leq i \leq c$. Then Q_1 is the only child of P in $\mathcal{T}'$, and thus $(R = P_1, \ldots, P_\ell = P, Q_1)$ is the maximal non-branching path of $\mathcal{T}'$ which has length $\ell + 1$. So it suffices to show that $t(\mathcal{T}) \leq t(\mathcal{T}')$.

Consider any activation sequence $A = (a_1, \ldots, a_f)$ where a_f is the first activation after which all particles in $\mathcal{T}$ have finished recharging; we must show that there exists an activation sequence A' such that $t_A(\mathcal{T}) \leq t_{A'}(\mathcal{T}')$. We construct A' from A so that the flow of energy through $\mathcal{T}'$ mimics that of $\mathcal{T}$. Consider each $a_i \in A$, for $1 \leq i \leq f$. In most cases, a_i has the same effect in both $\mathcal{T}$ and $\mathcal{T}'$ and thus $a'_i = a_i$ can be appended to A'. However, any activations a_i in which P passes energy to a child Q_j, for $2 \leq j \leq c$, cannot be performed directly in $\mathcal{T}'$ since Q_j is a child of Q_1 — not of P — in $\mathcal{T}'$. We instead add a pair of activations $a'_i = (a^1_i, a^2_i)$ to

[2] The *depth* of a particle P in a tree $\mathcal{T}$ rooted at a particle R is the number of nodes in the R, P-path in $\mathcal{T}$ (i.e., the root R is at depth 1, and so on). The depth of a tree $\mathcal{T}$ is $\max_{P \in \mathcal{T}}\{\text{depth of } P\}$.

A' that have the effect of passing energy from P to Q_j but use Q_1 as an intermediary. There are two cases. If Q_1 has a full battery (i.e., $Q_1.e_{bat} = \kappa$) at the beginning of a_i, then Q_1 passes energy to Q_j in a_i^1 and P passes energy to Q_1 in a_i^2. Otherwise, P passes energy to Q_1 in a_i^1 and Q_1 passes energy to Q_j in a_i^2.

Under this construction of A', if all particles start with empty batteries, then the value of $P.e_{bat}$ after each $a_i \in A$ and $a_i' \in A'$ is the same in $\mathcal{T}$ and $\mathcal{T}'$, respectively, for all $1 \le i \le f$. Thus, the particles in $\mathcal{T}$ and $\mathcal{T}'$ only finish recharging after activations a_f and a_f', respectively. Moreover, A' was obtained from A by adding activations which can only increase the number of asynchronous rounds in A'. Therefore, we have $t_A(\mathcal{T}) \le t_{A'}(\mathcal{T}')$, as desired. □

By Lemma 2.3, it suffices to analyze the case where $\mathcal{T}$ is a simple path of k particles. To bound the recharge time in this setting, we use a *dominance argument* between asynchronous and parallel executions which is structured as follows. First, we prove that for any asynchronous execution, there exists a parallel execution that makes at most as much progress towards recharging the system in the same number of rounds. We then upper bound the recharge time in parallel rounds. Combining these results gives a worst case upper bound on the recharge time in asynchronous rounds, as desired.

Let a configuration C of the path $P_1, \ldots, P_k$ encode the battery values of each particle P_i as $C(P_i)$. A *schedule* is a sequence of configurations $(C_0, \ldots, C_t)$. Note that in the following definition for the parallel execution, we reduce each particle's battery capacity from κ to $\kappa' = \kappa - \alpha$. This does not apply to the asynchronous execution, and is just a proof artifact that will be useful in Lemma 2.5.

Definition 2.4. A *parallel energy schedule* $(C_0, \ldots, C_t)$ is a schedule such that for all configurations C_i and particles P_j we have $C_i(P_j) \in [0, \kappa']$ and, for every $0 < i \le t$, C_i is reached from C_{i-1} using the following for each particle P_j:

- P_j is a root, so it harvests energy from the external energy source with $C_i(P_j) = C_{i-1}(P_j) + \min\{\alpha, \kappa' - C_{i-1}(P_j)\}$.
- $C_{i-1}(P_j) \ge \alpha$ and $C_{i-1}(P_{j+1}) < \kappa'$, so P_j passes energy to its child with:
  - $C_i(P_j) = C_{i-1}(P_j) - \min\{\alpha, \kappa' - C_{i-1}(P_{j+1})\}$
  - $C_i(P_{j+1}) = C_{i-1}(P_{j+1}) + \min\{\alpha, \kappa' - C_{i-1}(P_{j+1})\}$

Such a schedule is *greedy* if the above actions are taken in parallel whenever possible.

Now consider any fair asynchronous activation sequence A; i.e., one in which every particle is activated infinitely often. We compare a greedy parallel energy schedule to an *asynchronous energy schedule* $(C_0^A, \ldots, C_t^A)$ where C_i^A is the configuration of the path $P_1, \ldots, P_k$ at the completion of the i-th asynchronous round in A. For a particle P_i in a configuration C, let $\Delta_C(P_i)$ denote the total amount of energy in the batteries of particles $P_i, \ldots, P_k$ in C; i.e., $\Delta_C(P_i) = \sum_{j=i}^{k} C(P_j)$. For any two configurations C and C', we say C *dominates* C' — denoted $C \succeq C'$ — if and only if for all particles P_i in the path $P_1, \ldots, P_k$, we have $\Delta_C(P_i) \ge \Delta_{C'}(P_i)$.

Lemma 2.5. *Given any fair asynchronous activation sequence A beginning at a configuration C_0^A in which $P_i.e_{bat} = 0$ for all $1 \le i \le k$, there exists a greedy parallel energy schedule $(C_0, \ldots, C_t)$ with $C_0 = C_0^A$ such that $C_i^A \succeq C_i$ for all $0 \le i \le t$.*

Proof. Given a fair asynchronous activation sequence A and an initial configuration C_0^A, we obtain a unique asynchronous energy schedule $(C_0^A, \ldots, C_t^A)$. Our goal is to construct a parallel energy schedule $(C_0, \ldots, C_t)$ such that $C_i^A \succeq C_i$ for all $0 \le i \le t$. Let $C_0 = C_0^A$; then, for $0 < i \le t$, let C_i be obtained from C_{i-1} by performing one *parallel round*: each particle greedily performs the actions of Definition 2.4 if possible.

We now show $C_i^A \succeq C_i$ for all $0 \le i \le t$ by induction on i. Since $C_0 = C_0^A$, we trivially have $C_0^A \succeq C_0$. So suppose $i > 0$ and for all rounds $0 \le r < i$ we have $C_r^A \succeq C_r$. Considering any particle P_j, we have $\Delta_{C_{i-1}^A}(P_j) \ge \Delta_{C_{i-1}}(P_j)$ by the induction hypothesis and want to show that $\Delta_{C_i^A}(P_j) \ge \Delta_{C_i}(P_j)$. First suppose the inequality from the induction hypothesis is strict and we have $\Delta_{C_{i-1}^A}(P_j) > \Delta_{C_{i-1}}(P_j)$, meaning strictly more energy has been passed into $P_j, \ldots, P_k$ in the asynchronous setting than in the parallel one after rounds $i-1$ are complete. Because all successful energy transfers pass α energy either from the external source to the root P_1 or from a parent P_j to its child P_{j+1}, we have that $\Delta_{C_{i-1}^A}(P_j) \ge \Delta_{C_{i-1}}(P_j) + \alpha$. But by Definition 2.4, a particle can receive at most α energy per parallel round, so we have:

$$\Delta_{C_i}(P_j) \le \Delta_{C_{i-1}}(P_j) + \alpha \le \Delta_{C_{i-1}^A}(P_j) \le \Delta_{C_i^A}(P_j).$$

Thus, it remains to consider when $\Delta_{C_{i-1}^A}(P_j) = \Delta_{C_{i-1}}(P_j)$, meaning the amount of energy passed into $P_j, \ldots, P_k$ is exactly the same in the asynchronous and parallel settings after rounds $i-1$ are complete. It suffices to show that if P_j receives α energy in parallel round i, then it also does so in asynchronous round i.

We first prove that if P_j receives α energy in parallel round i, then $C_{i-1}^A(P_j) \le \kappa - \alpha$; i.e., P_j has enough room in its battery to receive α energy whenever it is activated in asynchronous round i. There are two cases: either P_j already had enough room in its battery to receive α energy in parallel round i (i.e., $C_{i-1}(P_j) \le \kappa' - \alpha$) or it had a full battery (i.e., $C_{i-1}(P_j) = \kappa'$) but passed α energy to P_{j+1} in parallel, "pipelining" energy to make room for the energy it received. In either case, it is easy to see that $C_{i-1}(P_j) \le \kappa'$. By supposition we have $\Delta_{C_{i-1}^A}(P_j) = \Delta_{C_{i-1}}(P_j)$ and by the induction hypothesis we have $\Delta_{C_{i-1}^A}(P_{j+1}) \ge \Delta_{C_{i-1}}(P_{j+1})$. These yield:

$$\begin{aligned}
C_{i-1}^A(P_j) &= \sum_{\ell=j}^{k} C_{i-1}^A(P_\ell) - \sum_{\ell=j+1}^{k} C_{i-1}^A(P_\ell) \\
&= \Delta_{C_{i-1}^A}(P_j) - \Delta_{C_{i-1}^A}(P_{j+1}) \\
&\le \Delta_{C_{i-1}}(P_j) - \Delta_{C_{i-1}}(P_{j+1}) \\
&= \sum_{\ell=j}^{k} C_{i-1}(P_\ell) - \sum_{\ell=j+1}^{k} C_{i-1}(P_\ell) \\
&= C_{i-1}(P_j) \le \kappa' = \kappa - \alpha
\end{aligned}$$

Thus, regardless of whether P_j already had space for α energy or used pipelining in parallel round i, P_j must have space for α energy at the start of asynchronous round i, as desired.

Next, we show that if P_j receives α energy in parallel round i, then there is at least α energy for P_j to receive in asynchronous round i. If P_j is the root, this is trivial: the external source of energy is its infinite supply. Otherwise, $j > 1$ and we can show

$C^A_{i-1}(P_{j-1}) \geq \alpha$ using the supposition $\Delta_{C^A_{i-1}}(P_j) = \Delta_{C_{i-1}}(P_j)$ and the induction hypothesis $\Delta_{C^A_{i-1}}(P_{j-1}) \geq \Delta_{C_{i-1}}(P_{j-1})$ in an argument analogous to the one above (see [6] for details).

Thus, we have shown that if P_j receives α energy in parallel round i, then $C^A_{i-1}(P_j) \leq \kappa - \alpha$ and either $j = 1$ or $C^A_{i-1}(P_{j-1}) \geq \alpha$, meaning that at the end of asynchronous round $i-1$ there is both α energy available to pass to P_j and P_j has room in its battery to receive it. Though we do not control the order of activations in asynchronous round i, additional activations can only increase the amount of energy available to pass to P_j (by, e.g., passing more energy to P_{j-1}) and increase the space available in $P_j.e_{bat}$ (by passing more energy to P_{j+1}). Since the activation sequence A was assumed to be fair, either $j = 1$ and P_j will be activated at least once in asynchronous round i or $j > 1$ and P_{j-1} will be activated at least once in asynchronous round i; in either case, P_j will receive α energy in asynchronous round i. Therefore, in all cases we have shown that $\Delta_{C^A_i}(P_j) \geq \Delta_{C_i}(P_j)$, and since the choice of P_j was arbitrary, we have $C^A_i \geq C_i$ as desired. □

To conclude the dominance argument, we bound the number of parallel rounds needed to recharge a path of k particles. Combined with Lemma 2.5, this gives an upper bound on the worst case number of asynchronous rounds required to do the same.

Lemma 2.6. *Let $(C_0, \ldots, C_t)$ be a greedy parallel energy schedule where C_0 is the configuration in which $P_i.e_{bat} = 0$ for all $1 \leq i \leq k$ and C_t is the configuration in which $P_i.e_{bat} = \kappa' = \kappa - \alpha$ for all $1 \leq i \leq k$. Then $t = \frac{\kappa'}{\alpha}k = O(k)$.*

Proof. If $k = 1$, then $P_1 = P_k$ is the root particle that harvests α energy per parallel round from the external source by Definition 2.4. Since P_1 has no children to which it may pass energy, clearly, within $\frac{\kappa'}{\alpha} = O(k)$ rounds $P_1.e_{bat} = \kappa'$ will be satisfied. Now suppose $k > 1$ and that for all $1 \leq j < k$, a path of j particles fully recharges in $\frac{\kappa'}{\alpha}j$ parallel rounds. Once a particle P_i has received energy for the first time, it is easy to see by inspection of Definition 2.4 that P_i will receive α energy from P_{i-1} (or the external energy source, in the case that $i = 1$) in every subsequent parallel round until $P_i.e_{bat}$ is full. Similarly, Definition 2.4 ensures that P_i will pass α energy to P_{i+1} in every subsequent parallel round until $P_{i+1}.e_{bat}$ is full. Thus, once P_i receives energy for the first time, P_i effectively acts as an external energy source for the remaining particles $P_{i+1}, \ldots, P_k$.

The root P_1 first harvests energy from the external energy source in parallel round 0, and thus acts as a continuous energy source for $P_2, \ldots, P_k$ in all subsequent rounds. By the induction hypothesis, we have that $P_2, \ldots, P_k$ will fully recharge in $\frac{\kappa'}{\alpha}(k-1)$ parallel rounds, after which P_1 will no longer pass energy to P_2. The root P_1 harvests α energy from the source per round and already has $P_1.e_{bat} = \alpha$, so in an additional $\frac{\kappa'}{\alpha} - 1$ parallel rounds we have $P_1.e_{bat} = \kappa'$. Therefore, the path $P_1, \ldots, P_k$ fully recharges in $1 + \frac{\kappa'}{\alpha}(k-1) + \frac{\kappa'}{\alpha} - 1 = \frac{\kappa'}{\alpha}k = O(k)$ parallel rounds, as required. □

Lemmas 2.3, 2.5, and 2.6 show that an inhibited tree $\mathcal{T}$ of k particles will recharge all its stressed particles in at most $O(k)$ asynchronous rounds. The following lemma shows that within a bounded number of additional rounds, there will be some particle that is neither inhibited nor stressed and thus can perform an enabled action (if it has one).

Lemma 2.7. *Suppose that the last stressed particle in $\mathcal{T}$ has just received the energy it needs to perform its next action. If $\mathcal{T}$ has depth $d_{\mathcal{T}}$, then within $2d_{\mathcal{T}}$ additional rounds some particle in $\mathcal{T}$ with a pending enabled action will be able to perform it.*

Proof. Let $\mathcal{T}_a$ be the set of particles in $\mathcal{T}$ that have enabled actions to perform. By supposition, all particles in $\mathcal{T}_a$ now have sufficient energy stored in their batteries to perform their actions (i.e., they are no longer stressed). It remains to bound the time for a particle in $\mathcal{T}_a$ to reset its inhibit flag, the only remaining obstacle to performing its action.

Let $\mathcal{S} \subseteq \mathcal{T}$ be the connected subtree of particles with their stress flags set. All leaves of $\mathcal{S}$ at the start of an asynchronous round are guaranteed to reset their stress flags by the completion of the round since they are no longer stressed and do not have children with stress flags set. A descendant-to-root path in $\mathcal{S}$ can have length at most $d_{\mathcal{T}}$; the depth of tree $\mathcal{T}$. So in at most $d_{\mathcal{T}}$ rounds, all particles in $\mathcal{T}$ will reset their stress flags.

In the first asynchronous round in which the root does not have any children with their stress flags set, the root resets its inhibit flag. In each subsequent round, any child whose parent has reset its inhibit flag will also reset its own inhibit flag. The longest root-to-descendant path in $\mathcal{T}$ is of length $d_{\mathcal{T}}$, so in at most $d_{\mathcal{T}}$ rounds there must exist a particle in $\mathcal{T}_a$ that resets its inhibit flag; let P be the first such particle. Particle P has an enabled action, has sufficient energy stored, and is not inhibited, so it performs its enabled action during its next usage phase. □

We conclude our analysis with the following two theorems. Recall from Section 1.3 that an algorithm solves the energy distribution problem in t asynchronous rounds if no particle remains stressed for more than t rounds and at least one particle is able to perform an enabled action every t rounds.

Theorem 2.8. *Algorithm Energy-Sharing solves the energy distribution problem in $O(n)$ asynchronous rounds.*

Proof. By Lemma 2.1, all n particles in system $\mathcal{P}$ will join the spanning forest $\mathcal{F}$ within n asynchronous rounds. Since there is no communication or energy transfer between different trees of $\mathcal{F}$, it suffices to analyze an arbitrary tree $\mathcal{T} \in \mathcal{F}$. By Lemma 2.2, if $\mathcal{T}$ contains a stressed particle then all particles of $\mathcal{T}$ will be inhibited within $2d_{\mathcal{T}}$ rounds, where $d_{\mathcal{T}}$ is the depth of $\mathcal{T}$. Lemma 2.3 shows that assuming $\mathcal{T}$ has a path structure can only increase the time to recharge its stressed particles, and Lemmas 2.5 and 2.6 prove that even in the case that all particles have uniform, maximum demand — i.e., $\delta(P) = \kappa$ for all particles P — all stressed particles will be distributed enough energy to meet their demand within $O(|\mathcal{T}|)$ rounds. Finally, Lemma 2.7 shows that within $2d_{\mathcal{T}}$ additional rounds some particle in $\mathcal{T}$ will use its energy to perform its next enabled action. Therefore, since the depth of $\mathcal{T}$ can be at most its size (if $\mathcal{T}$ is a path) and its size can be at most the number of particles in the system (if $\mathcal{T}$ is the only tree in $\mathcal{F}$), we conclude that Energy-Sharing solves the energy distribution problem in $n + 2d_{\mathcal{T}} + O(|\mathcal{T}|) + 2d_{\mathcal{T}} = O(n)$ asynchronous rounds. □

To establish a lower bound, observe that for a system of n particles each with a battery capacity of κ to fully recharge, the system needs to harvest and distribute $n\kappa$ total energy. Each particle with access to an external energy source may only be activated once per asynchronous round in the worst case. So in this worst case, a system with $s \leq n$ particles with energy access can harvest at most $s\alpha$ energy from external sources per asynchronous round. This yields the following theorem, demonstrating that our Energy-Sharing algorithm is asymptotically optimal when s is a fixed constant.

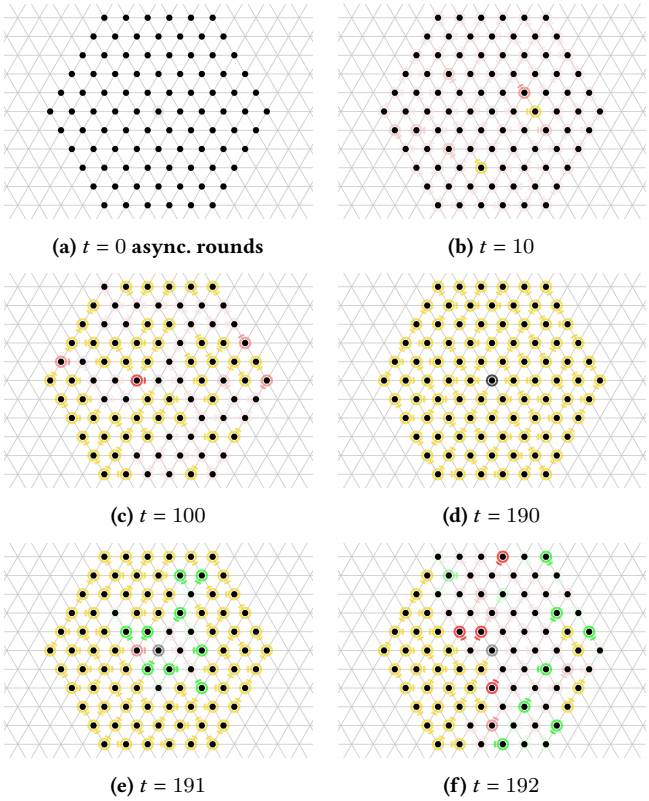
Theorem 2.9. *The worst case runtime for any local control algorithm to solve the energy distribution problem when $s \leq n$ particles have access to external energy sources is $\Omega(n/s)$ asynchronous rounds.*

3 SIMULATION RESULTS

We now present simulations of the Energy-Sharing algorithm.[3] All figures in this section use color intensity to indicate the energy level of a particle's battery, with more intense color corresponding to more energy stored. Our first simulation (Figure 2) shows Energy-Sharing running on a system of 91 particles with a single root particle that has access to an external energy source. All particles have a capacity of $\kappa = 10$ and a transfer rate of $\alpha = 1$. To incorporate energy usage in the simulation, we assume that every particle has a uniform, repeating demand of $\delta(\cdot, \cdot) = 5$ energy per "action", though no explicit action is actually performed when the energy is used. The system is organized as a hexagon with the root at its center for visual clarity, but the resulting behavior is characteristic of other initial configurations, root placements, and parameter settings.

All particles are initially idle, with the exception of the root shown with a gray/black ring (Figure 2a). The setup phase establishes the spanning forest (or tree, in this case) rooted at particle(s) with energy access; a particle's parent direction is shown as an arc. Since all particles start with empty batteries, stress flags (shown as red rings) quickly propagate throughout the system and inhibit flags soon follow (Figure 2b). As energy is harvested by the root and shared throughout the system, some particles (shown with yellow rings) receive sufficient energy to meet the demand for their next action but remain inhibited from using it (Figure 2c). This inhibition remains until all stressed particles in the system receive sufficient energy to meet their demands (Figure 2d), at which point particles (shown with green rings) can reset their inhibit flags and use their energy (Figure 2e). After using energy, these particles may again become stressed and trigger another stage of inhibition (Figure 2f).

Our second simulation demonstrates the necessity of the communication phase for effective energy distribution. In Section 1, we motivated the need for a strategy that leverages the biofilm-inspired long-range communication of particles' energy states to shift between selfish and altruistic energy usage. Figure 3 shows a simulation with the same initial configuration and parameters as the first simulation (Figure 2), but with its communication phase disabled. Without the communication phase to inhibit particles from using energy while those that are stressed recharge, particles continuously share any energy they have with their descendants in the spanning forest. Thus, while the leaves of the spanning forest occasionally meet their energy demands (bold green particles in Figure 3b–3d), even after 1000 rounds most particles have still not met their energy demand even once.



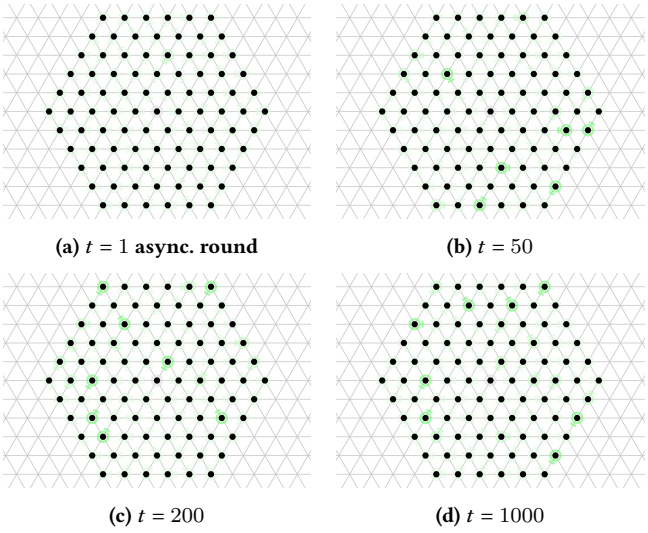
(a) $t = 0$ **async. rounds** **(b)** $t = 10$ **(c)** $t = 100$ **(d)** $t = 190$ **(e)** $t = 191$ **(f)** $t = 192$

Figure 2: A simulation of Energy-Sharing on 91 particles with one root, $\kappa = 10$, $\alpha = 1$, and a repeating uniform demand of $\delta(\cdot, \cdot) = 5$ for all particles. The black particle is the root, red particles have their stress flags and possibly also their inhibit flags set, yellow particles have only their inhibit flags set, and green particles have no flags set.

4 EXTENSIONS

With our energy distribution algorithm in place, we now present useful extensions. We begin by considering particle *crash failures* in which a particle stops functioning and no longer participates in the collective behavior. Crash failures pose a key challenge for Energy-Sharing: they disrupt the structure of the spanning forest $\mathcal{F}$ that the particles use for routing energy and communicating their energy states. To achieve robustness to these crash failures, we present algorithm Forest-Prune-Repair that enables the spanning forest to self-repair so long as certain assumptions on the locations of faulty particles hold (Sections 4.1–4.2). We then show how Forest-Prune-Repair can be leveraged to compose Energy-Sharing with existing algorithms in the amoebot catalogue, effectively generalizing all previous work on the amoebot model to also consider energy constraints (Section 4.3).

[3] Code for all simulations is openly available as part of AmoebotSim (https://github.com/SOPSLab/AmoebotSim), a visual simulator for the amoebot model of programmable matter. Enlarged videos of simulations can be viewed at https://sops.engineering.asu.edu/sops/energy-distribution.



(a) $t = 1$ **async. round** **(b)** $t = 50$

(c) $t = 200$ **(d)** $t = 1000$

Figure 3: A simulation of Energy-Sharing with the same initial configuration and parameters as in Figure 2, but with the communication phase disabled. Without communication to set stress and inhibit flags, all particles remain uninhibited (green), but only the leaves of the spanning forest ever amass enough energy to meet their demands.

We make three assumptions about crashed particles. First, the neighbors of a crashed particle can detect that it is crashed. Second, the subgraph induced by the positions of non-crashed particles must remain connected at all times; otherwise, there may be no way for components of non-crashed particles to communicate. Third, there must always be at least one non-crashed root particle; otherwise, the system would lose access to all external energy sources. We do not claim that these *detection*, *connectivity*, and *root-reliability* assumptions are necessary for fault tolerance, but each addresses a non-trivial challenge that is beyond the scope of this work.

4.1 The Forest-Prune-Repair Algorithm

In the context of our energy distribution algorithm, crash failures partition the spanning forest $\mathcal{F}$ into "non-faulty" trees $\mathcal{F}^*$ that are rooted at particles with energy access and "faulty" trees $\mathcal{F}'$ that are disconnected from any external energy source. Together, $\mathcal{F}^* \cup \mathcal{F}'$ form a forest that spans all non-crashed particles. To make our algorithms robust to these faults, we present Forest-Prune-Repair, a local, ad hoc reconstruction that self-repairs $\mathcal{F}$ to reform a spanning forest of trees rooted at particles with energy access.

Algorithm Forest-Prune-Repair works as follows. When a particle P finds that its parent has crashed, it knows it has become the root of a faulty tree in $\mathcal{F}'$. In response, P broadcasts a "prune signal" throughout this new tree by setting a *prune flag* in each of its children's memories, informing its descendants of the crash failure. It then clears its parent pointer, resets all flags, and becomes idle. Any particle that has its prune flag set does the same, effectively dissolving the faulty tree. Idle particles then rejoin an existing tree in a manner similar to the setup phase described in Section 2.1. When activated, an idle particle P considers all its root or active neighbors that do not have their prune flag set. Of these particles, P chooses one to be its parent in a round-robin manner; i.e., if P is ever pruned again, it chooses the next such particle to be its parent.

Integrating Forest-Prune-Repair with Energy-Sharing is straightforward. In the setting where the system is subject to crash faults, Forest-Prune-Repair simply replaces the setup phase described in Section 2.1. A particle proceeds with the communication, sharing, and usage phases if it is not idle and its parent is not crashed.

4.2 Analysis

We now analyze Forest-Prune-Repair, beginning with two straightforward lemmas that we state without proofs due to space constraints. Proofs can be found in [6]; the proof of Lemma 4.2 uses techniques identical to those of Lemma 2.2.

Lemma 4.1. *If a non-faulty tree $\mathcal{T} \in \mathcal{F}^*$ is initially acyclic, then under Forest-Prune-Repair it will remain acyclic. Moreover, there will always be at least one tree in $\mathcal{F}^*$.*

Lemma 4.2. *Suppose a particle crashes, yielding a new faulty tree $\mathcal{T} \in \mathcal{F}'$. For any particle P at depth d in tree $\mathcal{T}$, P will be pruned (i.e., set its children's prune flags, clear its memory, and become idle) in at most d asynchronous rounds.*

Under Forest-Prune-Repair, a pruned particle P chooses its new parent Q from among its root or active neighbors that do not have their prune flags set. There are two cases: (1) Q is in a non-faulty tree, meaning P has rejoined $\mathcal{F}^*$ as desired, or (2) Q is in a faulty tree, say $\mathcal{T} \in \mathcal{F}'$. In the latter case, there must be prune flags propagating throughout $\mathcal{T}$ because $\mathcal{T} \in \mathcal{F}'$, so Lemma 4.2 shows P will now be pruned again, this time from $\mathcal{T}$.

An especially bad version of this case would occur if a particle continually rejoined the tree it is pruning by choosing one of its descendants as its new parent. In fact, if this choice is not made carefully, it is possible that such a particle would always choose a descendant as its parent and thus never rejoin $\mathcal{F}^*$. We refer to this situation as a *chase cycle* due to the way the prune flag propagation "chases" the rejoining particles. However, since particles choose their new parents from among their eligible neighbors in a round-robin manner, chase cycles cannot continue for long:

Lemma 4.3. *Suppose a particle P in faulty tree $\mathcal{T} \in \mathcal{F}'$ has at least one neighbor in a non-faulty tree of $\mathcal{F}^*$. Then the number of times P will be pruned before it rejoins $\mathcal{F}^*$ is at most 6.*

Proof. Each time P is pruned, it chooses a new parent from among its active or root neighbors that do not have their prune flags set. By supposition, P has at least one such neighbor in a tree of $\mathcal{F}^*$. Moreover, its neighbor(s) in $\mathcal{F}^*$ will always be in the set of eligible new parents since every particle in a non-faulty tree is either a root or is active and is never pruned. By Lemma 4.2, P will be pruned again each time it chooses a parent in a faulty tree of $\mathcal{F}'$. In a round-robin selection, P can choose each neighbor in $\mathcal{F}'$ as its parent at most once before choosing a parent in $\mathcal{F}^*$, as desired. Every particle has at most 6 neighbors, so in the worst case the number of times P will be pruned before it rejoins $\mathcal{F}^*$ is 6. □

We conclude by bounding the stabilization time of Forest-Prune-Repair, which captures the time required for all particles to rejoin non-faulty trees once the last crash failure has occurred. We note

that our bound does not directly depend on the number of crash failures f, but rather on the number of non-crashed particles m removed from non-faulty trees as a result of the crash failures.

Theorem 4.4. *Suppose $f < n$ particles crash (where $n = |\mathcal{P}|$), yielding faulty trees $\mathcal{F}'$. If no other particles crash, all $m = |\mathcal{F}'|$ non-crashed particles rejoin $\mathcal{F}^*$ in $O(m^2)$ rounds in the worst case.*

Proof. If $m = 1$, then by Lemma 4.1 and the connectivity assumption all non-crashed neighbors of the single non-crashed particle $P \notin \mathcal{F}^*$ must be in $\mathcal{F}^*$. By Lemma 4.2, P will be pruned in one round; P will then choose a neighbor in $\mathcal{F}^*$ as its new parent in its next activation. So P rejoins $\mathcal{F}^*$ in at most $O(1) = O(m^2)$ rounds.

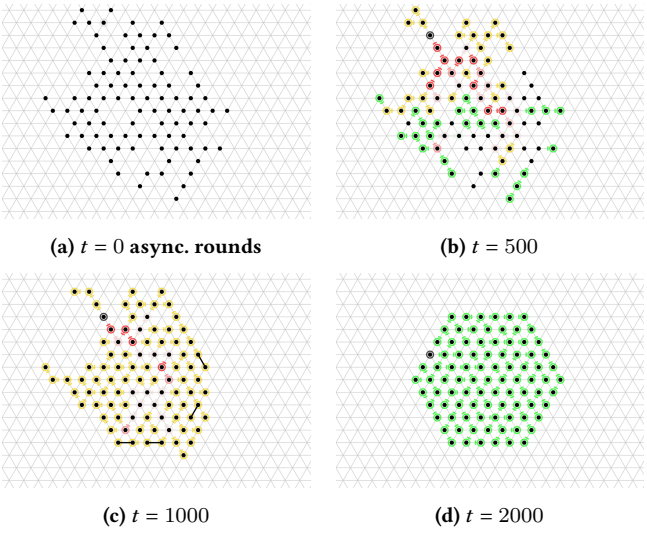
Now suppose $m > 1$. Again by Lemma 4.1 and the connectivity assumption, there must exist a non-crashed particle $P \in \mathcal{F}'$ with a neighbor in $\mathcal{F}^*$. By Lemma 4.2, P will be pruned in at most m rounds since the depth of P in its faulty tree can be at most the total number of particles in faulty trees. Particle P will then choose a new parent from among its eligible neighbors; if it chooses any neighbor in $\mathcal{F}'$ as its new parent, it will again be pruned in at most another m rounds by Lemma 4.2. By Lemma 4.3, P will in the worst case need to repeat this process 6 times before choosing a neighbor in $\mathcal{F}^*$ as its new parent. Thus, P rejoins $\mathcal{F}^*$ in $O(m)$ rounds. This leaves $m - 1$ non-crashed particles in $\mathcal{F}'$ needing to rejoin $\mathcal{F}^*$. By the induction hypothesis, these particles rejoin $\mathcal{F}^*$ in $O((m-1)^2)$ rounds, so we conclude that all m non-crashed particles in $\mathcal{F}'$ will rejoin $\mathcal{F}^*$ in $O((m-1)^2) + O(m) = O(m^2)$ rounds. □

4.3 Algorithm Composition

We ultimately envision Energy-Sharing as a subprocess that is executed continuously, handling the energy demands of higher level algorithms for the system's self-organizing behaviors. In particular, if every action of an amoebot algorithm was assigned an energy cost, Energy-Sharing must supply each particle with sufficient energy to meet these costs. However, many amoebot algorithms involve particle movements that would necessarily disrupt the spanning forest $\mathcal{F}$ maintained by Energy-Sharing for energy routing and communication. Just as was the case for crash failures (Section 4.1), this necessitates a protocol for repairing $\mathcal{F}$ as particles move, disconnecting from existing neighbors and gaining new ones.

We can repurpose Forest-Prune-Repair to address moving particles with a simple modification. In this setting, instead of a particle initiating the pruning of its subtree if it detects that its parent has crashed, it initiates the pruning of its subtree and additionally prunes itself (unless it is an energy root) whenever it moves according to the higher level algorithm. The rest of Forest-Prune-Repair stays the same with the pruning broadcast dissolving the subtree and the resulting idle particles rejoining elsewhere.

With this modification in place, Energy-Sharing can be composed with any amoebot algorithm $\mathcal{A}$ requiring energy distribution so long as (1) the battery capacity κ is at least as large as the demand of the most energy-intensive action in $\mathcal{A}$, and (2) $\mathcal{A}$ maintains system connectivity at all times (this is sufficient to satisfy the connectivity assumption of Section 4 since no particles actually crash). Note that $\mathcal{A}$ need not satisfy the root-reliability assumption; since each root is not actually crashing when it moves, the system maintains its access to external energy sources so long as it remains connected.



(a) $t = 0$ **async. rounds** **(b)** $t = 500$

(c) $t = 1000$ **(d)** $t = 2000$

Figure 4: A simulation of basic shape formation on 91 particles composed with Energy-Sharing with one root, $\kappa = 10$, $\alpha = 1$, and action demand $\delta(\cdot,\cdot) = 5$. The communication structure is maintained by Forest-Prune-Repair. Particle color and parent directions are visualized with respect to Energy-Sharing. The energy root moves according to the shape formation algorithm and need not be centered.

Actions required by algorithm $\mathcal{A}$ are handled in the usage phase of Energy-Sharing. If some particle P has an action to perform according to algorithm $\mathcal{A}$, then if P has sufficient stored energy and is not inhibited, it spends the energy and performs the action; otherwise, it foregoes its action this activation. For example, Figure 4 shows Energy-Sharing composed with the algorithm for *basic shape formation* [5, 8] forming a hexagon. Theorem 2.8 ensures that all n particles will meet their energy needs and at least one particle will be able to perform an enabled action every $O(n)$ asynchronous rounds. By Theorem 4.4, any disruption to the communication structure caused by actions involving movements will be repaired in $O(m^2)$ asynchronous rounds, where m is the number of particles severed from the communication structure. Thus, Energy-Sharing will not impede the progress of $\mathcal{A}$ but — according to our proven bounds — may add significant overhead to its runtime. However, we observe reasonable performance in practice: for example, since hexagon formation terminates in $O(n)$ rounds, our proven bounds suggest that the composed algorithm could terminate in time $O(n^2)$ or worse but Figure 5 demonstrates an overhead that appears asymptotically sublinear. With the addition of more energy roots, the composed algorithm is dramatically faster, approaching the runtime achieved without energy constraints [6].

5 CONCLUSION

In this work, we extended the amoebot model to include energy considerations. Our bacterial biofilm-inspired algorithm for energy distribution is guaranteed to meet the energy demands of a system of n particles at least once every $O(n)$ asynchronous rounds and is asymptotically optimal when the number of external energy

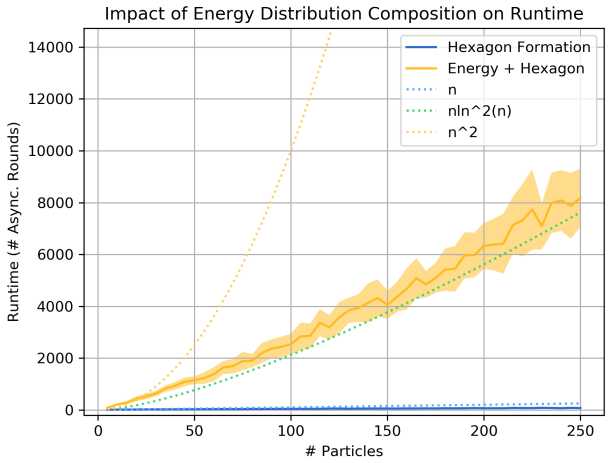


Figure 5: Runtimes of the basic shape formation algorithm alone (blue) vs. when it is composed with Energy-Sharing (yellow) as a function of system size. Each experiment was repeated 20 times; average runtime is shown as a solid line and standard deviation is shown as an error tube. Asymptotic runtime bounds are shown as dotted lines; the composed algorithm tracks most closely with $O(n \log^2 n)$.

sources is fixed. Existing amoebot model algorithms satisfying some basic assumptions can be generalized to respect energy constraints through composition with our energy distribution and spanning forest repair algorithms. Moreover, the spanning forest repair algorithm will be independently useful for future work in addressing fault tolerance for existing amoebot model algorithms.

Our goal in this work was to meet the energy demands of fixed-sized particle systems as they execute algorithm actions. One could also consider using energy for *reproduction*, mimicking the bacterial biofilms that inspired our algorithm. In preliminary simulations [6], we obtain behavior that is qualitatively similar to the biofilm growth patterns observed by Liu and Prindle et al. [17, 24]; in particular, the use of communication and inhibition leads to an oscillatory growth rate. Further work is needed to formally characterize our algorithm's behavior for these growing, dynamic systems.

ACKNOWLEDGMENTS

The authors gratefully acknowledge their support from the National Science Foundation under awards CCF-1637393 and CCF-1733680 and from the U.S. Department of Defense under MURI award #W911NF-19-1-0233. We thank Prof. Deborah Gordon and Prof. Saket Navlakha for their pointers to research on bacterial biofilms and their helpful discussions that initiated this work. We also thank Prof. Theodore Pavlic for generously sharing his knowledge of bio-inspired approaches to energy management in swarm robotics. Finally, we thank undergraduate researcher Christopher Boor for his contributions to a preliminary version of this work.

REFERENCES

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

[2] Palina Bartashevich, Doreen Koerte, and Sanaz Mostaghim. 2017. Energy-saving decision making for aerial swarms: PSO-based navigation in vector fields. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI '17)*. 1–8.

[3] Philip Bernstein, Vassos Hadzilacos, and Nathan Goodman. 1987. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

[4] Jason Campbell, Padmanabhan Pillai, and Seth Copen Goldstein. 2005. The Robot is the Tether: Active, Adaptive Power Routing for Modular Robots With Unary Inter-robot Connectors. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '05)*. IEEE, 4108–4115.

[5] Joshua J. Daymude, Kristian Hinnenthal, Andréa W. Richa, and Christian Scheideler. 2019. Computing by Programmable Particles. In *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*. 615–681.

[6] Joshua J. Daymude, Andréa W. Richa, and Jamison W. Weber. 2020. Bio-Inspired Energy Distribution for Programmable Matter. (2020). Available online at https://arxiv.org/abs/2007.04377.

[7] Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. 2014. Brief Announcement: Amoebot - A New Model for Programmable Matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '14)*. 220–222.

[8] Zahra Derakhshandeh, Robert Gmyr, Thim Strothmann, Rida Bazzi, Andrea W. Richa, and Christian Scheideler. 2015. Leader Election and Shape Formation with Self-organizing Programmable Matter. In *DNA Computing and Molecular Programming*. 117–132.

[9] Shlomi Dolev, Sergey Frenkel, Michael Rosenblit, Ram Prasadh Narayanan, and K Muni Venkateswarlu. 2016. In-vivo energy harvesting nano robots. In *2016 IEEE International Conference on the Science of Electrical Engineering*. 1–5.

[10] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro (Eds.). 2019. *Distributed Computing by Mobile Entities*. Springer International Publishing, Switzerland.

[11] Kyle Gilpin, Ara Knaian, and Daniela Rus. 2010. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *2010 IEEE International Conference on Robotics and Automation (ICRA '10)*. 2485–2492.

[12] Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Fabian Kuhn, Dorian Rudolph, and Christian Scheideler. 2018. Shape Recognition by a Finite Automaton Robot. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS '18)*. 52:1–52:15.

[13] Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Fabian Kuhn, Dorian Rudolph, Christian Scheideler, and Thim Strothmann. 2019. Forming tile shapes with simple robots. *Natural Computing* (2019).

[14] Seth Copen Goldstein, Jason D. Campbell, and Todd C. Mowry. 2005. Programmable Matter. *Computer* 38, 6 (2005), 99–101.

[15] Serge Kernbach. 2013. *Handbook of Collective Robotics: Fundamentals and Challenges*. Jenny Stanford Publishing.

[16] Sam Kriegman, Douglas Blackiston, Michael Levin, and Josh Bongard. 2020. A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences* 117, 4 (2020), 1853–1859.

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

[18] Bruce J. MacLennan. 2015. The Morphogenetic Path to Programmable Matter. *Proc. IEEE* 103, 7 (2015), 1226–1232.

[19] Othon Michail, George Skretas, and Paul G. Spirakis. 2019. On the transformation capability of feasible mechanisms for programmable matter. *J. Comput. System Sci.* 102 (2019), 18–39.

[20] Sanaz Mostaghim, Christoph Steup, and Fabian Witt. 2016. Energy Aware Particle Swarm Optimization as search mechanism for aerial micro-robots. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI '16)*. 1–7.

[21] Nils Napp, Samuel Burden, and Eric Klavins. 2011. Setpoint regulation for stochastically interacting robots. *Autonomous Robots* 30, 1 (2011), 57–71.

[22] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron, and Magnus Egerstedt. 2017. The Robotarium: A remotely accessible swarm robotics research testbed. In *2017 IEEE International Conference on Robotics and Automation (ICRA '17)*. 1699–1706.

[23] Benoit Piranda and Julien Bourgeois. 2018. Designing a quasi-spherical module for a huge modular robot to create programmable matter. *Autonomous Robots* 42 (2018), 1619–1633.

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

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

[26] Hongxing Wei, Bin Wang, Yi Wang, Zili Shao, and Keith C.C. Chan. 2012. Staying-alive path planning with energy optimization for mobile robots. *Expert Systems with Applications* 39, 3 (2012), 3559–3571.

[27] Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. 2013. Active Self-Assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science (ITCS '13)*.