KU LEUVEN

Faculty of Engineering Science
Master in Artificial Intelligence
ECS Option
Academic year 2019-2020

Simulation of Artificial Bio-Nanobots for Cell Repair / Protection Using Swarm of Kilobots

By Sid Mahapatra

June 2020

Simulation of Artificial Bio-Nanobots for Cell Repair / Protection Using Swarm of Kilobots

Sid Mahapatra
Multi Agent Systems
Master of Artificial Intelligence

June 02, 2020

Abstract

This paper simulates the behavior of Cell Rovers, the cell sized bio-nanobots envisioned to be injected into Astronauts, by using a swarm of Kilobots. Alone, each Kilobot is unable to perform any meaningful function, but as a group they perform emergent behavior matching that of as Cell Rover built using nanomaterials called Dendrimers. The simulation captures the searching, movement, and protective behavior of a Cell Rover, but fails to simulate the repair of the damaged bodies. The paper explores and applies themes found in Particle Swarm Optimization, Ant Colony Optimization, and the Wolfpack Algorithm. It is found that the concept underlining the function of a Cell Rover is similar to a general optimization problem. The algorithm implemented simulates the function of a Cell Rover, but could also be used as an optimization algorithm where the strength of a chemical gradient across a search space represents the objective function. An example of such an application is to develop assembler nanobots that can actively repair damaged Mars-suit by operating as space-filling polyhedrons.

Contents

1 Introduction 4
1.1 Motivations 4

2 Swarm Intelligence Algorithms 5
2.1 The Optimization Problem 5
2.2 Particle Swarm Optimization 6
2.3 Ant Colony Optimization 8
2.4 Wolfpack Algorithm 9

3 Cell Rovers 10

4 Software 11
4.1 Swarm Robotics 11
4.2 Kilobots 12
4.3 Description of Problem 12
4.4 Implementation 13
4.4.1 Gradient 13
4.4.2 Movement 16
4.4.3 Protection 18
4.4.4 Repair 19
4.5 Kilombo 20
4.6 Results 22

5 Conclusion 27

References 29

List of Figures

1 Surface plot of $\frac{\sin(5x)\cos(5y)}{x}$ [6] 6
2 Ant Colony Optimization Simulation Using Unity 8
3 Process of Cell Crawling [1] 11
4 Example Kilobot [4] Alongside College of Wooster Kilobots 12
5 Light Reading Propogation in Kilobot Swarm. 16
6 Flowchart a Kilobot's Behavior 20
7 Screenshot of Kilombo Simulator. 21
8 Kilombo Simulation With Single Target 23
9 Kilobots Independently Move to Light With Single Light Source. 24
10 Kilobots Independently Move to Light With Two Light Sources. 25

11 Kilombo Simulation With Two Targets. 26
12 Scalability and Adaptability of Radbot.......... 27

List of Algorithms

1 Basic Gradient Setting 14
2 Adaptive Gradient Setting.......... 15
3 Kilobot Movement.......... 18

1 Introduction

Humans are built on a centralized intelligence paradigm; our brain is an individual authority that controls most things happening in our body, from the mundane bodily functions to critical thought. As a result, most of modern computing is built on the same paradigm. Consider, for example, that the ever-present Von Neumann architecture has a central processing unit. Or, in the case of networking, a fundamental structure is the client-server structure, in which a centralized server sends information to clients.

Decentralized intelligence, while less relatable for humans is another approach to intelligence in organisms and systems. This form of intelligence is best exemplified by social animals like ants, bees, and birds. Instead of individual actors making their own decisions or following some dedicated central authority, these groups act as a collective. In an ant colony for example, all ants follow a set of simple rules to determine their activity. They sense their surrounding environment and can tell what their neighbors are doing, and adjust their behavior accordingly. Alone, they are incapable of anything impressive, but, as a group or swarm, can perform complex, emergent behaviors like building a colony.

In 1986, Craig Reynolds created one of the first algorithms simulating natural swarm behavior when he wrote his Boids algorithm, which models the flocking of birds. Each agent in the algorithm represents a bird which moves by combining three simple rules: avoid neighbors, steer towards the average heading of all nearby birds, and steer towards the average position of all nearby birds. Following, his work, the field of swarm intelligence was initiated, in which the collective behavior of natural and artificial systems using decentralized intelligence are studied. For the purposes of this paper, however, swarm intelligence is considered in terms of artificial systems [7].

Since the 1980s, swarm intelligence algorithms have been used in a plethora of applications. This paper explores a simulation of Cell Rovers, an important biological feature (under development) for Astronauts, by applying swarm intelligence strategies to a group of robotic agents. It first describes two famous swarm intelligence algorithm families as well as the more recent Wolfpack Algorithm before exploring the mechanics behind Cell Rovers. Finally, it defines the approach and implementation of the simulation environment used to model it, followed by the results of the simulation.

1.1 Motivations

Decentralized intelligence has many advantages compared to centralized intelligence models. Swarms tend to not have a central point of failure; destroying a swarm agent will not have a huge impact on the performance of the swarm.

Swarms are robust and scalable. Most, if not all, of the agents follow similar sets of simple rules, so it is easy to add agents to a swarm and immediately have them start contributing to any emergent behavior [2].

Finally, because of an agent's simplicity, its performance cost, or manufacturing cost, in the case of physical swarm agents, can be much lower than centralized-intelligence style autonomous agents.

These attributes make swarm intelligence applicable to describe the behavior of near-horizon technology like Cell Rovers. The tiny artificial cell-like vessel tasked with finding and repairing damaged cells as witnessed in a scene from the movie *Innerspace,* acts as an inspiration for this work. Human cells are constantly dying and being created, and certainly don't possess any real centralized intelligence or thought. Instead, groups of cells cooperate to perform more complex functions in higher order structures like organs. Thus, any endeavor to simulate such a biological function benefits from a reliance on swarm intelligence; the pursuit of which might yield improvement on current swarm intelligence methods or insight into medical mechanisms.

2 Swarm Intelligence Algorithms

Before discussing Cell Rovers in depth, this paper delves into optimization and swarm intelligence algorithms used to perform optimization, since the topic has relevance to the way Cell Rovers behave.

2.1 The Optimization Problem

Optimization - the search for solutions to an optimization problem - is critical to any machine learning model. When analytical methods can't be used to find an optimal weight vector for a model (e.g. the weight vector of linear regression can be calculated), an optimization method is required to find a good weight vector. In any optimization problem, there is a search space defined by a set of variables, and an objective function that, given values for each of the unknowns, will compute a quantity that the optimization technique seeks to minimize or maximize.

For example, Figure 1 shows the surface for an example function, $\frac{\sin(5x)\cos(5y)}{x}$. The plot's surface represents the value of the function for any pair of unknowns, x and y. This particular objective function has an infinite number of global optima, since each bump shares the same maximum, and each dip shares the same minimum. With a slight amount of random jitter, this optimization problem could be much more complex since there would only be one global maxima and minima and many other local optima.

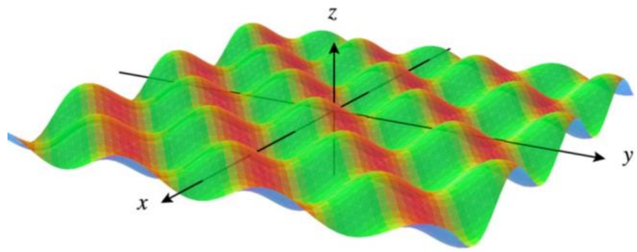

Figure 1: Surface plot of $\frac{\sin(5x)\ \cos(5y)}{x}$ [6]

One traditional optimization algorithm is gradient descent. In its simplest form, gradient descent first picks random values for each of the unknowns. Then, it calculates the partial derivative of the unknowns with respect to each unknown.

The resulting vector is the direction in which the gradient descent will step and then repeat the process. After this, it continues to take steps in the direction of the minimum objective value.

Gradient descent, however, is not without its problems. It can be slow to converge to an optimum, and can zig-zag towards the optimum depending on the shape of the objective function. In addition, it's difficult to efficiently set the step size; too high, and the gradient descent will oscillate stepping over the optimum, too low, and it will never get there in time. Finally, it is hard to search the entire space effectively. If the objective function has many local optima, then the gradient descent approach might easily get stuck in one near its initialization [3].

Swarm optimization algorithms can offer many advantages to traditional optimization techniques. Because of this, and the high demand for optimization algorithms, two algorithm families have come to dominate the swarm intelligence field: ant colony optimization (ACO) and particle swarm optimization (PSO).

2.2 Particle Swarm Optimization

PSO is an optimization technique used for searching a continuous search space. In every PSO algorithm, a swarm of particles blankets the search space, each of which is capable of calculating the value of the objective function at its position (i.e. its set of unknowns). The algorithm iterates through a series of discrete time steps, during which, the positions of the agents change. In each time step, a velocity is added to the current position of a particle, giving it a new position in the search space. This velocity is a combination of their previous velocity (initialized randomly at the start), a cognitive component, and a social component.

The cognitive component relates to a particle's memory; each particle keeps track of the location in which it has had the best objective function value. The cognitive component is proportional to the distance the particle currently is from that location. Thus, if the particle is far from its memorized best location, this has a large impact on the velocity that repositions the particle.

The social component is the optimum location across the particle's entire neighborhood (or the whole swarm if global PSO is used, see below). Similar to the cognitive component, the social component is proportional to the distance the particle currently is from that location. Thus, if the particle is far from the neighborhood's best location, this has a large impact on the velocity that repositions the particle. In order to make the algorithm stochastic, the social component and cognitive component are each multiplied by a random value between 0 and 1.

This is the basic structure of any PSO algorithm, but over time many variations have come to exist. One of the most important modifications is the way in which particles communicate. For example, if each particle communicates with every other particle, in what is called a star social structure, then each particle always knows what the global best position is; its neighborhood is the whole swarm. This method converges very quickly, but can easily be trapped in local optima; each member of the swarm is drawn to the same location. If the global best happens to be in a local optimum, that's where the swarm will gather. Thus, it's best for applications with a single optimum.

Under another structure, the ring, each particle can only communicate with a finite number of nearest neighbors. Thus, multiple (overlapping) neighborhoods exist, each with their own local optima. This slows down the propagation of information among the swarm, effectively reversing the drawbacks and benefits of the star structure. A swarm member's social component will draw it towards the optima of its neighborhood instead of the optima of the entire swarm.

In addition to changing the communication structure there are a number of modifications that exist. Velocity clamping bounds the value that a particle velocity can be, which prevents the velocities from exploding, as they often do. The previous velocity of a particle can also be multiplied by an inertia weight to increase or decrease its impact on the velocity update - an inertia weight > 1 causes particles to accelerate toward convergence, while < 1 causes them to decelerate.

Overall, PSO offers much different optimization structure compared to gradient descent. While it is impossible to tell when PSO has found an optimum, it can cover a much greater search area. One of the greatest benefits compared to gradient descent is that PSO can be used in a search space where it is impossible to calculate partial derivatives, since the velocity updates for each particle are based on distances alone [2].

2.3 Ant Colony Optimization

While PSO excels at continuous optimization problems, ACO is primarily used for discrete optimization problems. It also is more focused on probabilistic methods rather than velocities.

In an ant swarm, a common activity is foraging for food. Each ant moves toward the food by smelling the pheromones left by other ants on the surrounding ground. A simulation of this can be seen in Figure 2.

The figure depicts an ant colony foraging for food at three different time steps. The ants (black spheres) want to find the shortest of two paths from their home (blue square) to food (green square). In the initial time step, they randomly choose their path, with around half choosing the longer path.

Those ants on the shorter path return first, as seen in the second time step, and drop pheromones (green spheres) at that path's head. Once they've returned to their home, the ants probabilistically choose their path again, but this time with more weight on the shorter path. The third time step shows that, over time, the ants converge on the shortest path.

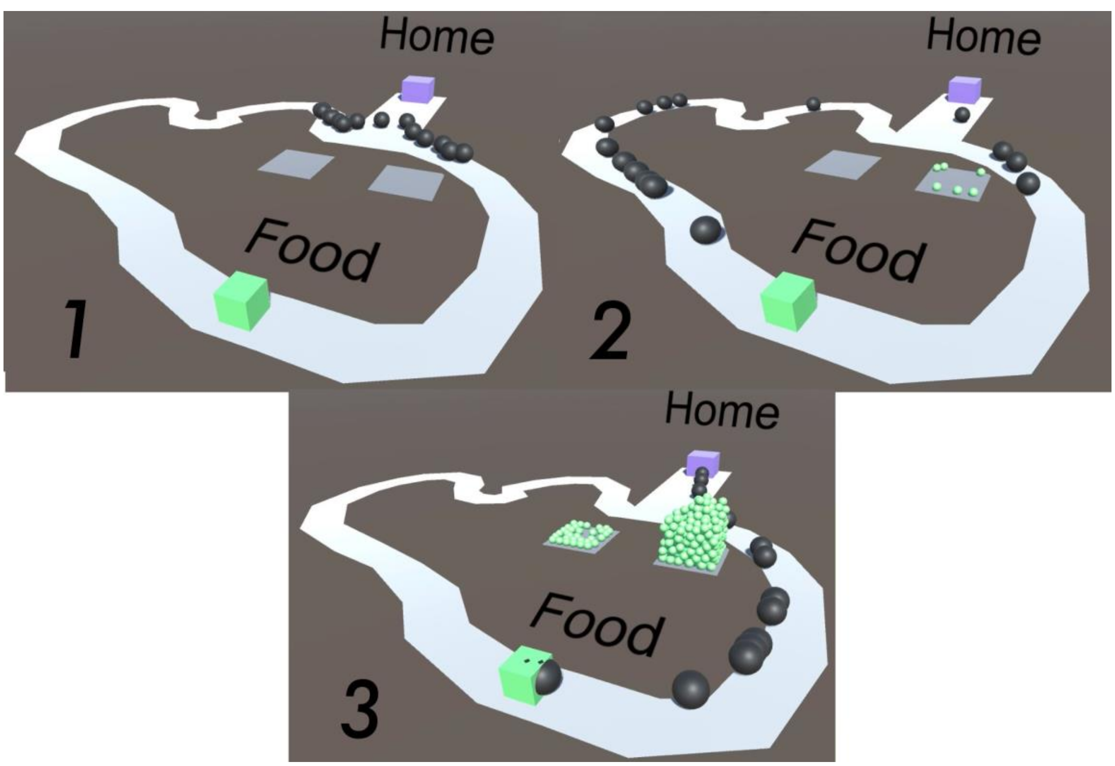

Figure 2: Ant Colony Optimization Simulation Using Unity

This behavior can be easily used to find shortest paths in graphs, and forms the basis for the simplest ACO algorithms. In these algorithms, agents in the swarm are analogous to ants. The swarm starts at a node in the graph and another node is designated as the food node. The edges from each node to another represent the paths the swarm agents can traverse. Each agent chooses a path based on the pheromone levels in the neighboring nodes.

Since the search space in this work is continuous, however, this is sufficient discussion on ACO [2].

2.4 Wolfpack Algorithm

A more recent addition in the world of swarm intelligence algorithms, the WPA borrows much more from PSO than ACO, and is used for continuous global optimization problems. The authors wrote the algorithm after they noticed the efficiency with which tundra wolves hunt. In it, each swarm agent represents a wolf, while the prey's scent is the objective function they seek to optimize (the prey's location is the highest scent concentration).

Unlike PSO or ACO, the WPA is not a homogeneous swarm - not all wolves are the same. The swarm agents can either be scouting wolves, ferocious wolves, besieging wolves, or the lead wolf. The swarm agent with the highest initial objective function becomes the lead wolf. This agent then calls all the other agents towards it, similar to a howling pack leader.

Other agents then enter the scouting behavior; they proceed to take steps in a random number of directions until they either find a higher objective function value (scent) than the lead wolf agent or hit a set number of max steps. If a scouting agent finds a higher objective function value, it becomes the new lead wolf. However, if it reaches the max number of scouting steps, it enters the ferocious state and moves towards the lead wolf, converging on the optimal value.

Agents conducting ferocious behavior keep moving towards the lead wolf until they are within a threshold distance (a parameter of the algorithm), at which point they move to the besieging behavior. In this final behavior, the agents continue taking steps towards the lead wolf while the steps increase their objective function value. If a step results in a lower objective function, they stop. When a besieging agent finds an objective function higher than the lead wolf agent, the prey is deemed 'found'. The agents with the lowest objective functions at this time are removed, and new agents are instantiated to take their place. In this way the algorithm simulates the winner-takes-all behavior of a wolf pack [11].

3 Cell Rovers

As stated earlier, a Cell Rover is built with nanomaterials called Dendrimers and might one day prove to be a crucial line of defense in an Astronaut's body against high-energy radiation. It is envisioned to be injected into Astronauts to repair or protect cells from radiation, <u>one cell at a time</u>. The artificial cells would use the body's cellular signaling system (chemical flags) to communicate with other cells, including its own kind, and to track damaged cells. They first surround the damaged cell in a closed circular pattern to protect it from further radiation damage. The second aspect to their function (not implemented in this work), is their ability to access the damage and take an appropriate action.

In order to find the damaged cell, Cell Rovers rely on chemotaxis - movement of a cell in response to a chemical gradient. Cell Rovers are like eukaryotic cells, which are large enough that they can detect a chemical gradient across their body; receptor proteins cluster on one side of the cell. Often, the damaged cell that a Cell Rover is looking for, releases a chemical creating a chemical gradient around them across which the Cell Rover travels [10].

While the chemotaxis might tell Cell Rovers *where* to go, it does not dictate their mechanism of motility - how they actually move. The method Cell Rovers can use to move across chemical gradients is one employed by many other animal cells: cell crawling (see Figure 3). In this method, the cell traverses a substrate based on a series of adhesion and deadhesion. Because protein receptors cluster at an edge of the cell as it follows a chemotaxis, Actin (a protein that helps maintain cellular shape) in the cytoskeleton of the cell polymerizes in this area, creating a protrusion. Adhesion molecules enter this protrusion and cause it to stick to the substrate. As this happens, the actin structure in the rear adhesion sites disassembles and the cell moves forward as the cytoskeleton contracts towards the adhesion sites [1].

When a Cell Rover comes into contact with the damaged cell, it binds onto it using more protein receptors. Other Cell Rovers surround the damaged cell in a closed circular pattern to protect it from further radiation damage. Based on the extent of the damage, the Cell Rovers would then either destroy it or release DNA-repair enzymes to fix it.

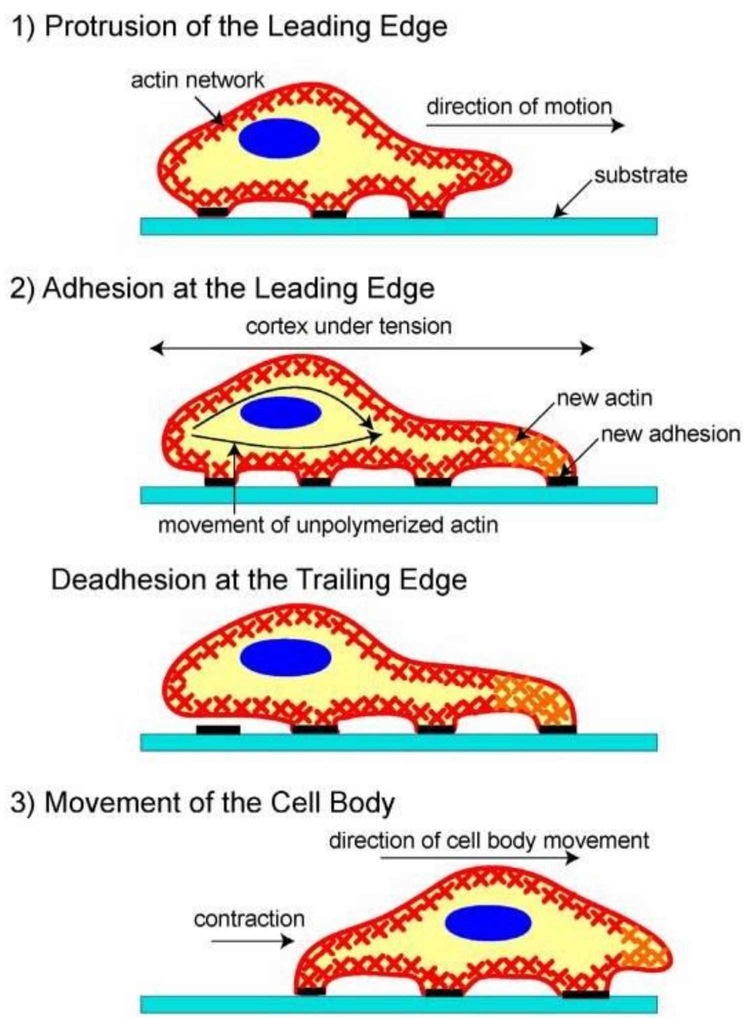

Figure 3: Process of Cell Crawling [1]

4 Software

Following the description of Cell Rovers as well as some algorithmic fundamentals that underly its simulation, this section explores the actual implementation of the simulation.

4.1 Swarm Robotics

The simulation uses a group of robots operating with a rule-set that is swarm-like. Swarm robotics, the intersection of swarm intelligence and robotics, can be used to describe any robotic system in which there are multiple, simple robots whose coordination and interaction with the environment results in emergent behaviors.

4.2 Kilobots

The agents used in the simulation are Kilobots, a low-cost robot with a robust set of features developed by researchers at Harvard (see Figure 4 for an image). Kilobots are capable of stick-and-slip movement; they have three metal legs, which, when vibrated, can make the Kilobot rotate in a direction or travel forward. In addition, a Kilobot has the ability to detect ambient light and temperature. Each Kilobot also possesses an RGB LED. Finally, they can broadcast an infrared message roughly twice every second. Any Kilobot within in a 7cm radius can receive these messages. The message itself is 12 bytes long, 9 of which are dedicated to carrying data.

Figure 4: Example Kilobot [4] Alongside College of Wooster Kilobots

Operating with an AVR microcontroller (AVR is a family of microcontrollers developed by Atmel), Kilobots have 32kb of flash memory. Any new program is written to this storage via an overhead IR broadcasting board. The programs for a Kilobot are written in C and compiled using the avr-gcc compiler, which outputs files in HEX format - an ASCII representation of binary [4].

4.3 Description of Problem

Having outlined the capabilities and specifications of the Kilobots, it is important to define the problem in terms of these. Behavior of Cell Rovers is a combination of global optimization and pattern formation. Cell Rovers traverse a search space (e.g. the bloodstream) trying to maximize an objective function (e.g. the concentration of a chemical released by a damaged cell). Once they find a damaged cell, they must completely surround it, which is a similar process to pattern formation. Thus, the Kilobots are responsible for finding some sort of target, and cutting it off from the rest of the search space.

With this in mind, there are two general approaches one could take. In the first, the Kilobots traverse the search space in a dispersed manner, only coming together when they converge on a target. At this point, they are responsible for forming an appropriate pattern to surround the target's area and close it off from the rest of search space. This approach looks similar in many respects to PSO, but only loosely resembles the Cell Rovers, since the Kilobots aren't behaving like cells [10]. In addition, the search space the Cell Rovers must traverse is quite vast. An applied version of the simulation requires an infeasible swarm size in order to effectively maintain swarm communication between neighborhoods.

The second approach, which this project pursues, involves replicating individual Cell Rovers and their behavior. In this approach, groups of Kilobots make up distinct Cell Rovers. While traversing the search space, groups of Kilobots attempt to stick together and follow the method of cell crawling that Cell Rovers use. For the remainder of the paper, these groups are referred to as Radbots. As above, once a Radbot gets close enough to a target, its member Kilobots need to communicate and move to surround the target.

The stick-and-slip movement of each Kilobot requires that the search space for a Radbot is a flat, 2D space (a whiteboard is an appropriate surface). There are three methods that could be used simulate a chemical gradient on such a search space given each Kilobots sensory abilities: light, heat, or a secondary, static network of Kilobot's broadcasting gradient values. The last of these introduces many problems (e.g physical collision between broadcasting Kilobots and Radbot Kilobots), and temperature changes relatively slowly, leaving light as the most reasonable choice. Thus, each Radbot must move towards increased light in the search space and surround the sources of such light.

4.4 Implementation

Here, the implementation of the major roles of a Radbot - recognizing a gradient, moving across a gradient, and protecting a target - are detailed.

4.4.1 Gradient

Each Radbot must be aware of the gradient of light along its body. This gradient is simulated by building upon existing gradient code for Kilobots [4]. In this pre-existing code, each Kilobot constantly broadcasts its own gradient value. A seed robot, who has a static gradient value of 0, is designated by manually calibrating its ID. All non-seed Kilobots grab gradient values from any incoming messages and set their gradient to the lowest received gradient incremented by 1. In this way Kilobots who receive a message from the seed Kilobot have a gradient value of 1, while those who can hear messages from these Kilobots, but not the seed have a gradient value of 2 (and so on). The gradient number essentially tells each Kilobot how many hops it is from the seed.

If a Kilobot doesn't hear a repeat of the the lowest gradient value it has received in the last two seconds, it forgets this gradient and resets it based on current incoming messages. Thus, after pulling a Kilobot away from the seed by a few hops and leaving it for two seconds, it adjusts its gradient value accordingly. This behavior can be seen in Procedure 1.

Procedure 1 Basic Gradient Setting

while true **do**
    if $ownID = seedRobotID$ **then**
        $ownGradient \leftarrow 0$
        continue
    $minGradient \leftarrow \infty$
    for all $neighbor \in NeighborList$ **do**
        if $neighbor.gradient < minGradient$ **then**
            $minGradient \leftarrow neighbor.gradient$
    $ownGradient \leftarrow minGradient + 1$

This model, however, is not enough for the Radbot. Here, to borrow from the terminology used in cell crawling, the seed robot represents the Radbot's adhesion site to the surface it traverses. Under the basic gradient setting, there is no way for this seed robot can change, since a Kilobot whose ID is chosen to be the seed ID, will always be the seed robot. The Radbot, however, needs a way to dynamically change which of its Kilobots is the seed robot from which the gradient is formed. When a Kilobot in the Radbot detects light higher than that of the current Kilobot acting as an adhesion site, it should become the new adhesion site.

In order to achieve this functionality, each Kilobot constantly collects light samples while broadcasting a message containing its gradient value, the highest light recording it knows about, its ID, and a time to live value. Under this setup, the maximum light recording in the entire Radbot will propagate throughout the Radbot and become known to each Kilobot. Then, a Radbot can make a decision about its gradient based on its own light recording compared to the highest known light recording. This behavior can be seen in Procedure 2. The propagation of the highest known light recording amongst the Kilobots is shown in Figure 5. The numbers from top to bottom are the Kilobot's own recording, highest known recording, and time to live value.

Procedure 2 Adaptive Gradient Setting

```
while true do
    ProcessIncomingMessages()
    Broadcast(highestKnownRecording, ID, ownGradient, TTL)
    if ownRecording is highestKnownRecording then
        ownGradient ← 0
        continue
    minGradient ← ∞
    for all neighbor ∈ NeighborList do
        if neighbor.gradient < minGradient then
            minGradient ← neighbor.gradient
    ownGradient ← minGradient + 1
```

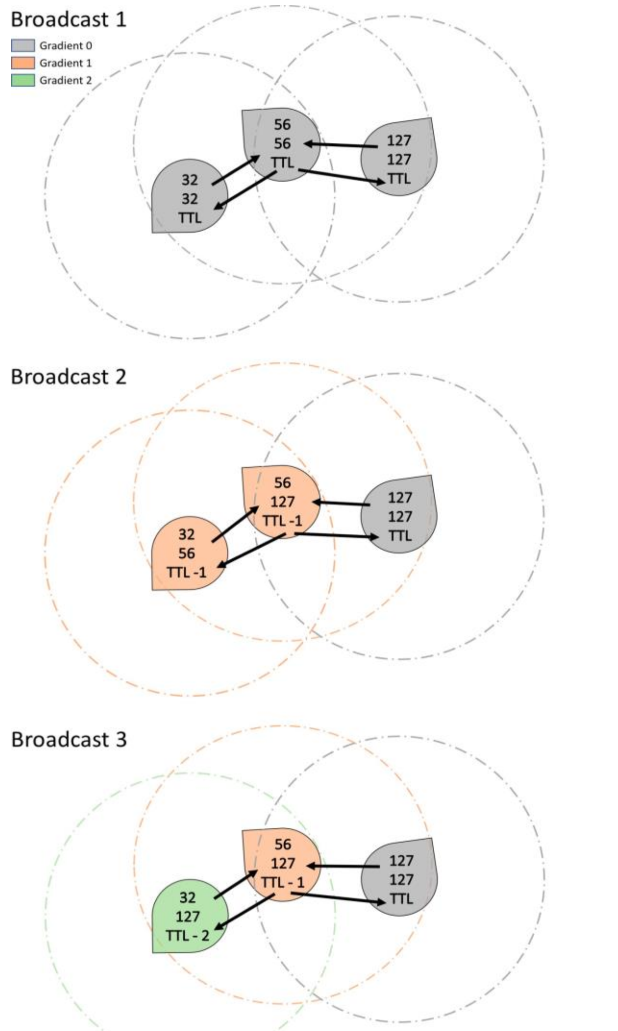

Figure 5: Light Reading Propagation in Kilobot Swarm.

4.4.2 Movement

While the Kilobot serving as the adhesion site determines the start of the light gradient across the Radbot, the other Kilobots must move towards it. This process is simulated by building upon pre-existing edge following code [5]. Since Kilobots have no sense of direction on their own, their movement is based on calculating their distance relative to other Kilobots and performing the appropriate behavior in relation to this - namely turning left or right, going forward, or stopping.

In the existing edge following code, one Kilobot moves along the edge of a group of static Kilobots who broadcast dummy messages so the mover can keep track of distances to its nearest neighbors. The moving Kilobot finds the nearest distance and turns right if it is below a certain threshold and left otherwise. Since it gains ground in the process of turning, it oscillates between turning left and right as it moves back and forth over this threshold, thereby proceeding along the edge.

Once again, this model, is not enough for the Radbot, since *each* Kilobot must move towards the one serving as the adhesion point. In order to simulate the contraction of a Cell Rover's cytoskeleton as it moves towards an adhesion point, the edge following code is changed to incorporate the gradient. Only, the Kilobots farthest away from the adhesion site Kilobot - in other words, those that form the 'back' of the Radbot - move at any given time. In addition, each Kilobot's orientation of left and right is randomized. In this way, not every Kilobot follows the edge of the Radbot in the same way. This prevents them from all swing around the adhesion Kilobot in the same direction.

At some point, a moving Kilobot following the edge moves beyond the adhesion site Kilobot, detects a higher light reading, and becomes the new adhesion site robot. In order for this new adhesion site to extend past the old, this new adhesion site Kilobot continues forward until the its nearest neighbor is far enough away. The movement behavior of the Kilobots can be seen in Procedure 3

Procedure 3 Kilobot Movement

```
leftMotor ← randomMotor
rightMotor ← otherMotor
while true do
    if ownGradient ≠ 0 then
        move ← true
        for all neighbor ∈ NeighborList do
            if (neighbor.isMoving) ∨ (neighbor.gradient > ownGradient) then
                move ← false
        if move then
            if distanceNearestNeighbor < threshold then
                moveRight()
            else
                moveLeft()
        else
            stopMove()
    else
        if distanceNearestNeighbor < commRange/1.25 then
            moveForward()
        else
            stopMove()
```

A final component to a Kilobot's movement behavior is momentum. When a Kilobot has no neighbors, it no longer attached to the Radbot. In order to reattach, it continues its current motion long enough to turn 180 degrees (with the default Kilobot turn speed, this is around 14 seconds). After this time, it moves forward until it reconnects (or doesn't as sometimes happens).

4.4.3 Protection

Kilobots lock into having a gradient value of 0 if they detect light above a certain threshold, even if this isn't the highest detected light of the swarm. This threshold is symbolic that it has reached the source of the light gradient and arrived at its target. In this manner, multiple Kilobots can be at gradient 0 once they reach the target. Other Kilobots will continue the movement behavior, following the edge until they too read light above the threshold. In this manner, Kilobots slowly travel around the edge of the light source, locking into place.

The value of the threshold determines how far away from the light source do the Kilobots begin encirclement of the target to protect it from further damage.

4.4.4 Repair

The Kilobots do not perform any repair or destructive behavior once the light source is encircled. They have no analog to a real Cell Rover's production and application of DNA-repair enzymes to repair the damaged cell.

The overall decision making of a Kilobot combines the gradient setting, movement, and protective behaviors described, is summarized in Figure 6.

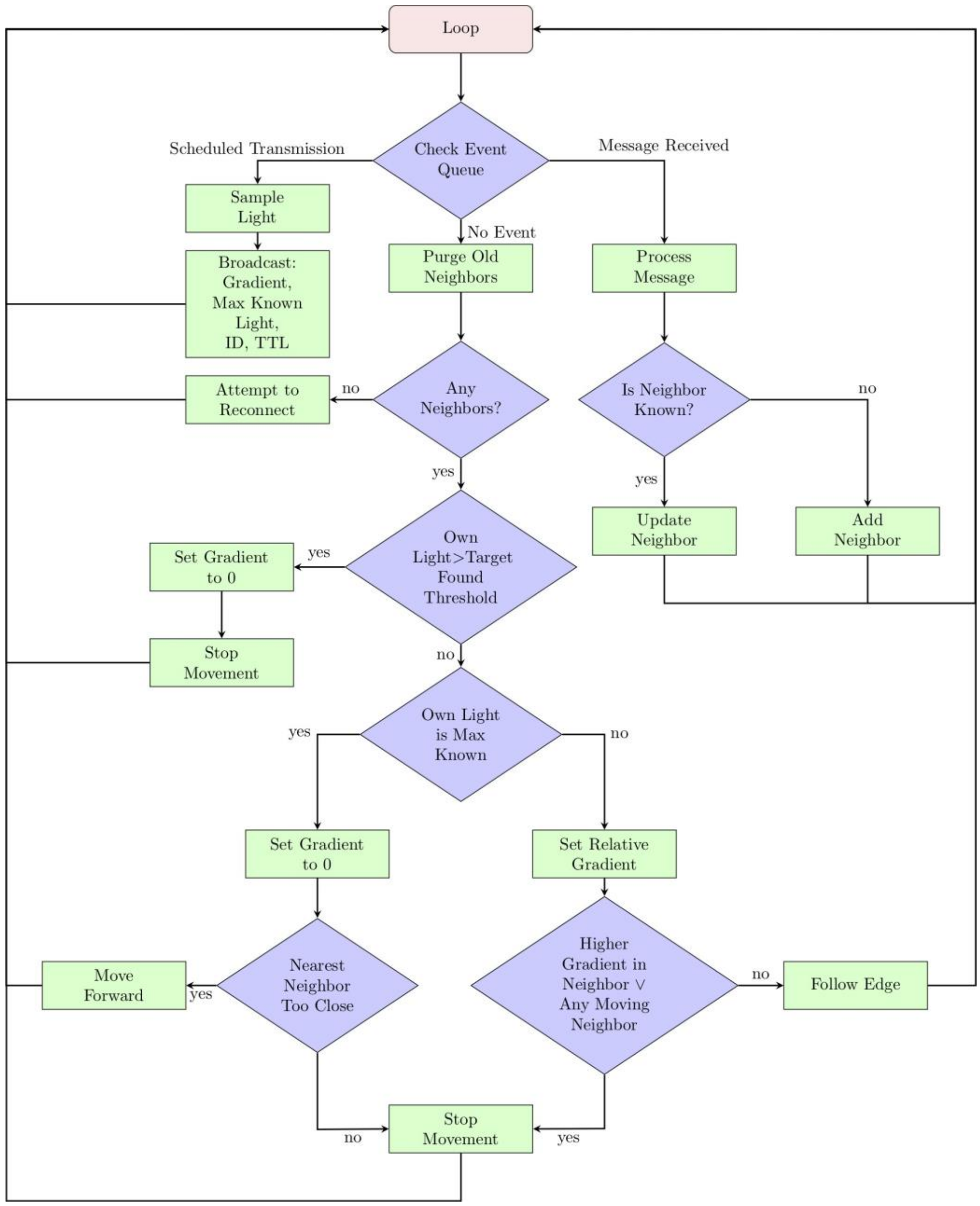

Figure 6: Flowchart a Kilobot's Behavior

4.5 Kilombo

It is important to note that during the implementation of the simulation, a change was made from using real Kilobots to using Kilombo, a Kilobots simulator written in C [5]. The change was made for two reasons.

The first of these was consistency and debugging purposes. Since Kilobots are physical robots, they require calibration, and the stick-and-slip movement style is not the most precise science. The Kilobots often need re-calibration, and only move consistently on the smoothest of surfaces. In addition, the only way to get output from the Kilobots was to use their serial output ports and plugging them directly into a computer. Kilombo Kilobots, however, offer a much more forgiving testing environment. A Kilobot turning left will actually turn left instead of angrily vibrating backwards off of the search space, and the program can be tested as soon as it is compiled. Debugging information from a Kilobot also appears inside the Kilombo GUI while the simulation is running.

Secondly, Kilombo easily allows for the addition of Kilobots to a simulation. No physical or DIY Kilobots were available for purchase due to the Covid-19 pandemic.

Several additional libraries are required to run the simulator, but a port of a native Kilobot program will look very similar to the native program.

Figure 7 shows an example of what the simulator looks like while running. The simulator is essentially a birds-eye view of a surface containing Kilobots. Each circle represents a single Kilobot, while the connecting lines show a Kilobot's neighbors. The color of the Kilobot is equivalent to its LED. The text at the top is metrics concerning the simulator, while the text on the bottom is debugging information for a single Kilobot.

The Kilombo simulator is not capable of rendering a light gradient, but can simulate one. Thus, for the rest of the paper, this light gradient is drawn using Blender tools to provide visualization.

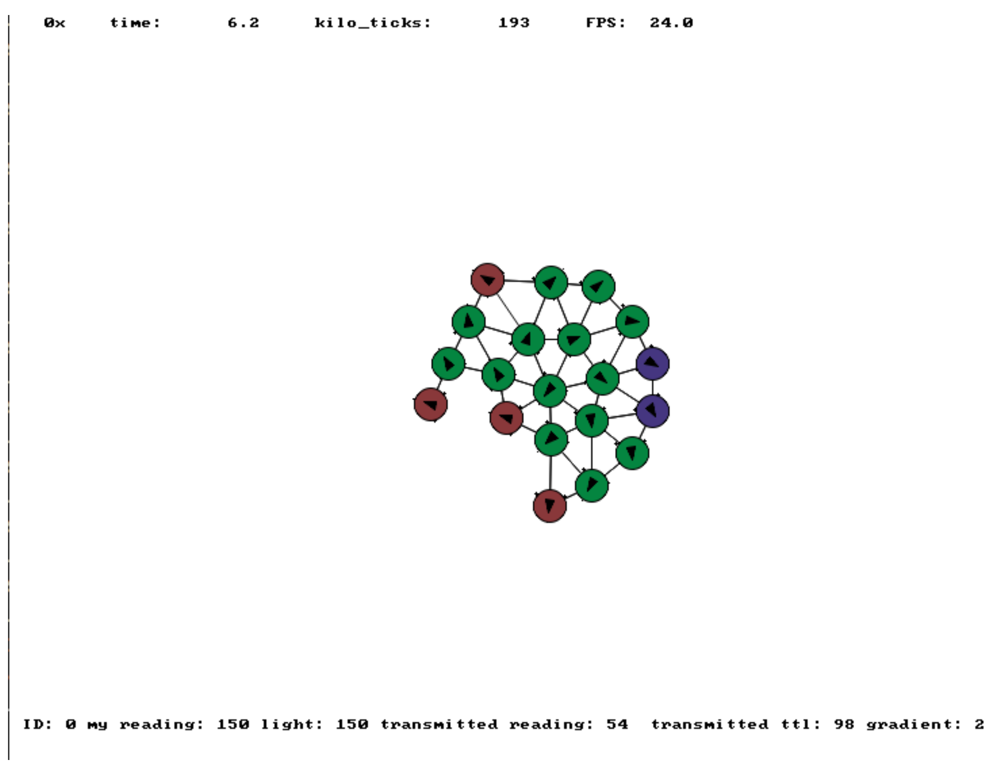

Figure 7: Screenshot of Kilombo Simulator.

4.6 Results

The implemented Cell Rover program is run in simulations using both a single light source and two light sources. The results of these simulations are compared to identical environments in which Kilobots independently move towards light. Before discussing these results, it is important to note that, in the simulations of Cell Rovers, Kilobots light up the following colors to indicate status:

1. **Blue -** The Kilobot has a gradient of 0. It could be an adhesion site, broadcasting its own light reading, or a Kilobot that has locked into place because its own light recording is above a set threshold.

2. **Green -** The Kilobot has a gradient > 0. However, it detects either moving neighbors, or neighbors with a higher gradient, so it doesn't move.

3. **Red -** The Kilobot has a gradient > 0. It detects no moving neighbors or neighbors with a higher gradient, so it follows the edge of the Radbot.

4. **Gray -**The Kilobot detects no neighbors, and will attempt to turn 180 degrees before proceeding forward, in an effort to reconnect with the Radbot.

The first simulation is conducted using a single light source and a swarm of 18 Kilobots running the Cell Rover program. A visualization of this simulation is shown in Figure 8. As stated previously, the gradient seen and depiction of a damaged cell are purely to aid the reader in observing the behavior, since the simulator cannot render the light gradient.

The visualization is split into four-time steps, each taken at roughly quarterly intervals of the simulation. In the first-time step, the maximum reading amongst all the Kilobots has propagated throughout the swarm, and several Kilobots are following the edge of the Radbot towards the adhesion site Kilobot.

The second and third-time steps show the Radbot's progress across the search space, while the fourth depicts their convergence around the light source. In this simulation, it took 28 minutes for the Radbot to surround the light source in a closed circle pattern.

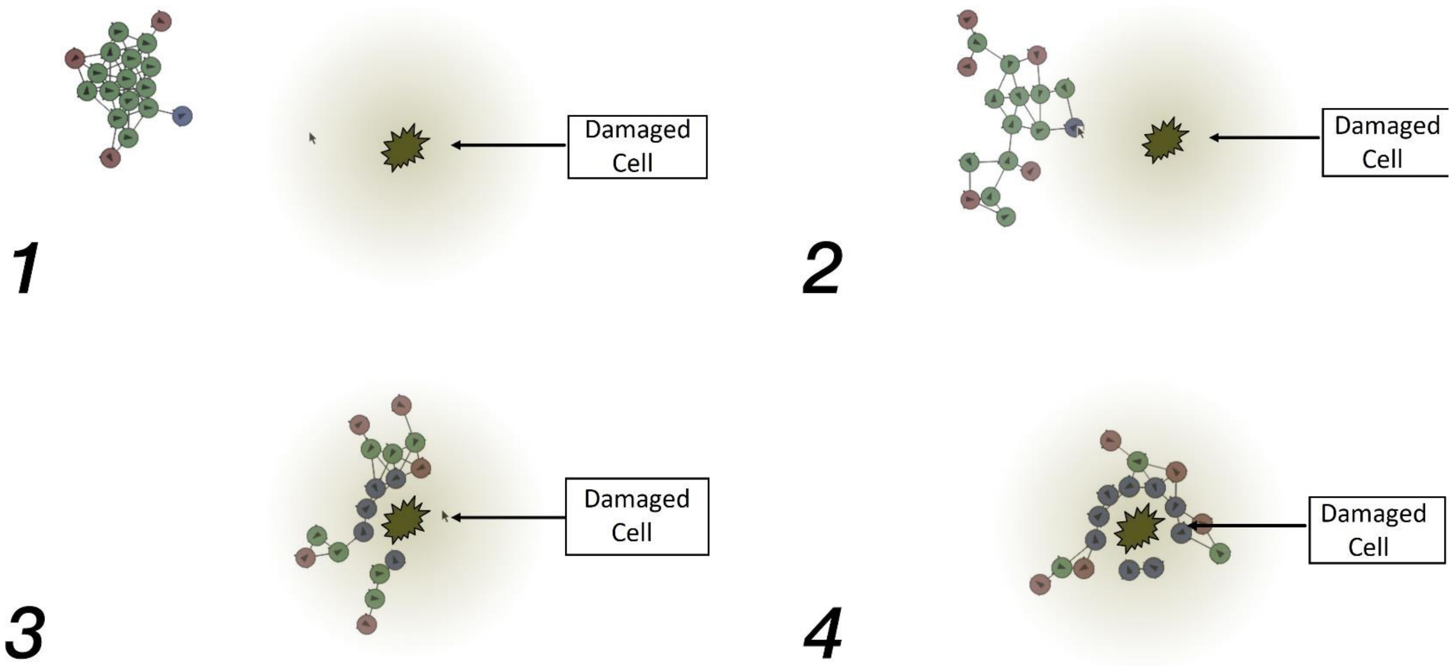

Figure 8: Kilombo Simulation with Single Target

The next simulation features a swarm of 18 Kilobots and single light source each in the same spot as the previous simulation. In this simulation, however, the Kilobots are moving towards the light independent of one another.

A visualization of this is shown in Figure 9. Again, the visualization is split into roughly quarterly intervals. Here, the Kilobots converged much more quickly on the light, only taking 5 minutes.

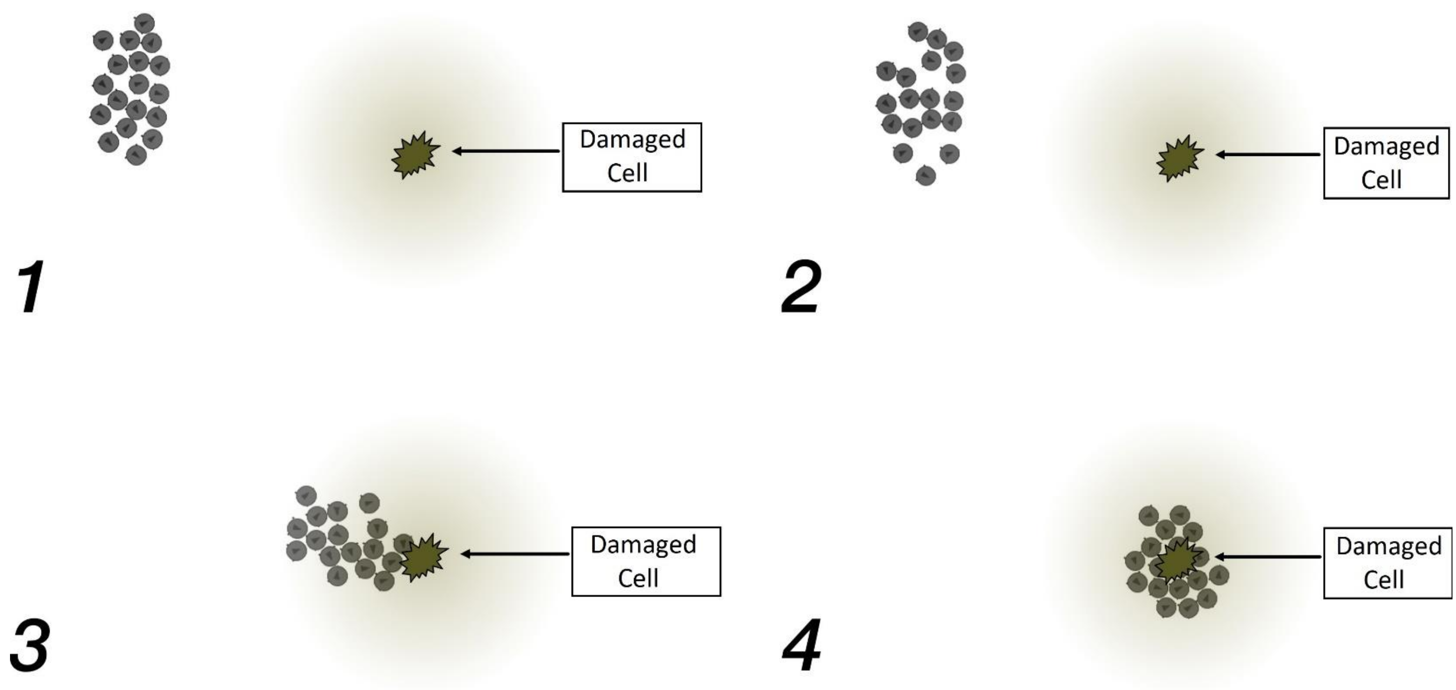

Figure 9: Kilobots Independently Move to Light with Single Light Source.

Despite the increase in speed, a critical flaw is revealed in this approach when more than one light source is considered. Figure 10 shows a visualization of a simulation in which 18 Kilobots independently move towards light in a search space featuring two lights. Again, the visualization is split into roughly quarterly intervals.

Here, the swarm splits, as Kilobots converge on both lights. One could imagine a search space with many more lights, where each Kilobot moves towards a unique light.

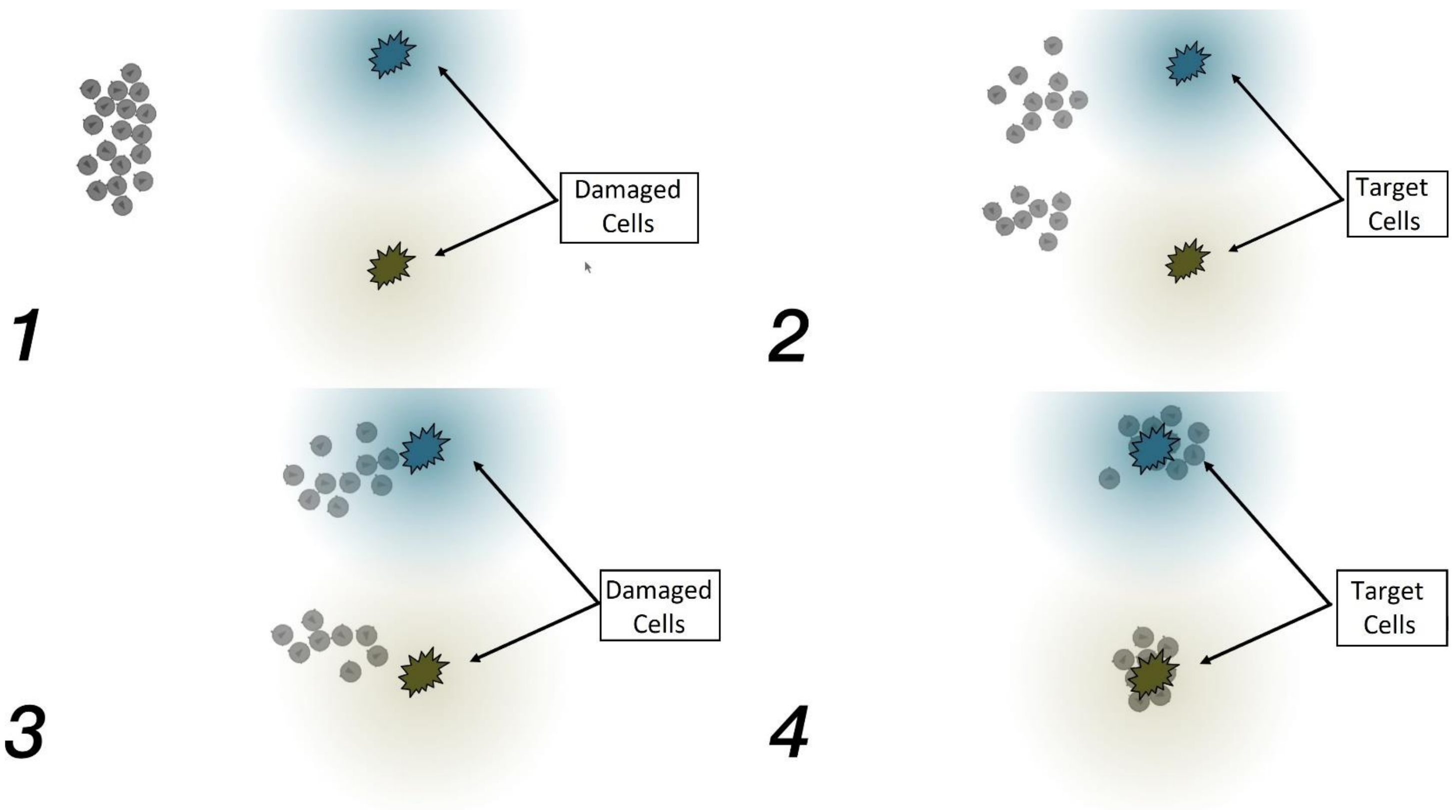

Figure 10: Kilobots Independently Move to Light with Two Light Sources.

In the same search space, however, a swarm of 18 Kilobots running the program is shown to converge on a single light source (see Figure 11). Because there is usually only one adhesion site Kilobot in a Radbot (there can be more than one if Kilobots share a light recording of the same value), the rest of the Kilobots are drawn to that Kilobot. Thus, the Kilobots in the Radbot are drawn to a single source.

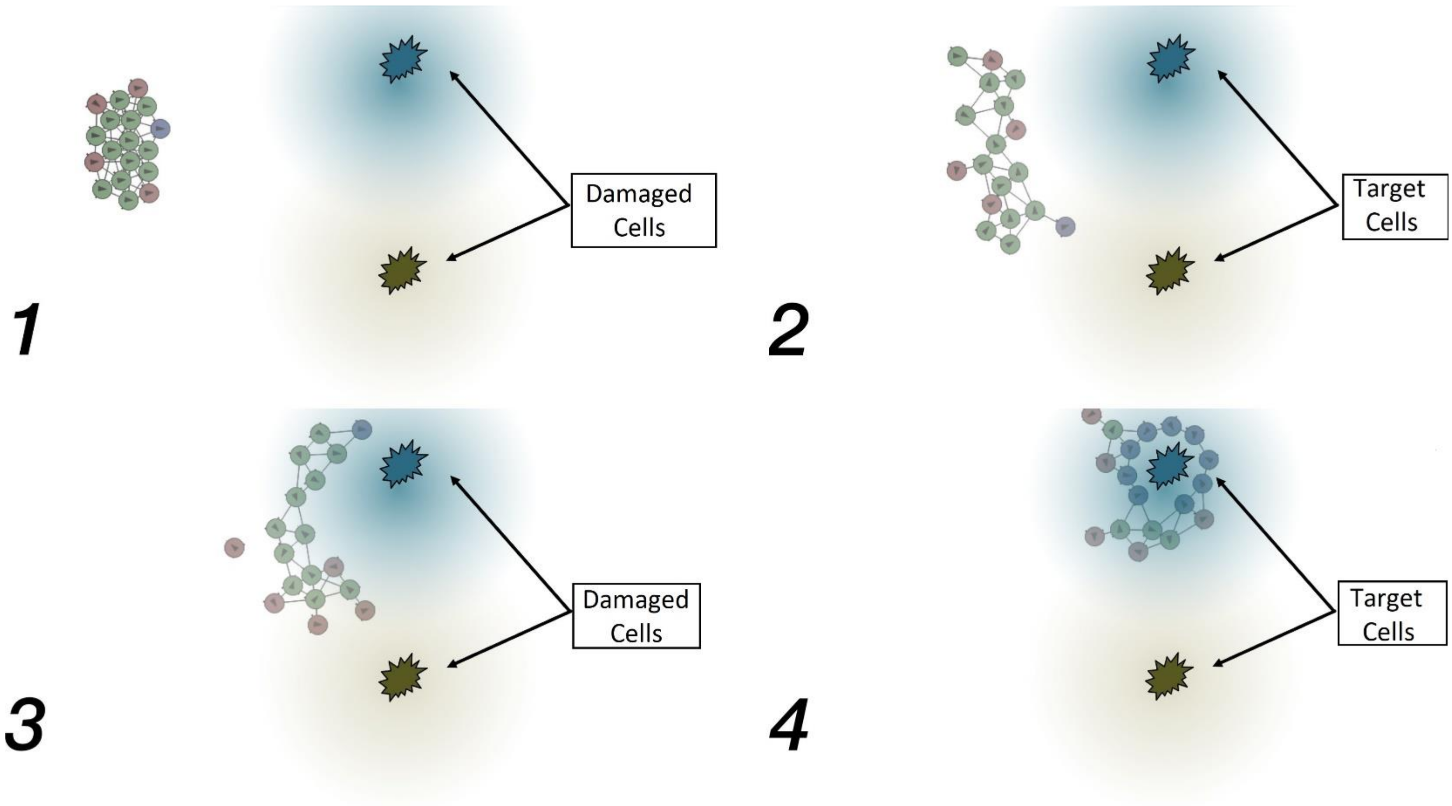

Figure 11: Kilombo Simulation With Two Targets.

Having compared those four simulations, Figure 12 shows the adaptability and scalability of a swarm of Kilobots running the simulation program. It is divided into four frames. The first shows a healthy swarm. In the second, the swarm loses its adhesion site Kilobot. Confusion ensues, but because of the time to live values being broadcasted in the swarm, a new adhesion site is chosen in frame 3. Frame 4 shows a Kilobot being added to the swarm and successfully integrating.

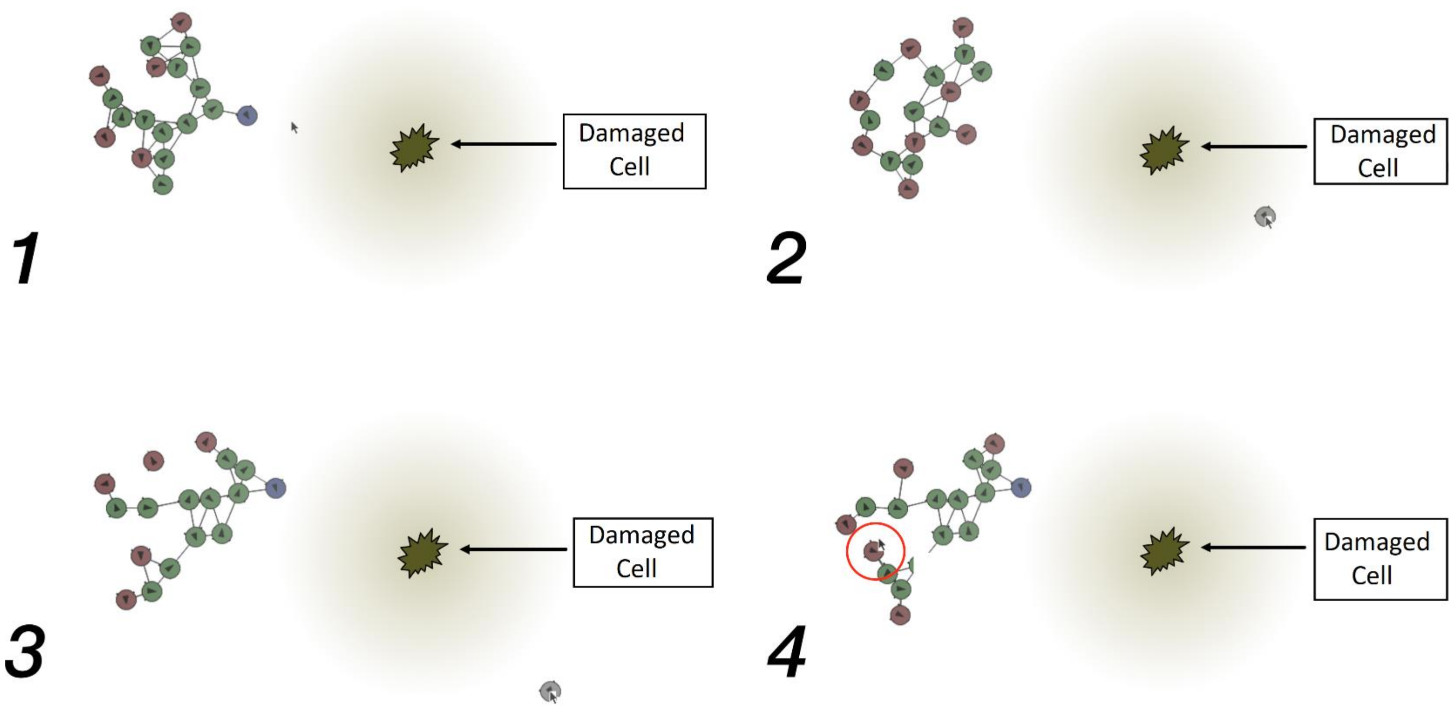

Figure 12: Scalability and Adaptability of Radbot.

5 Conclusion

As of right now, the implemented program successfully simulates Cell Rover's behavior using a swarm of Kilobots up to the protection of the damaged cell. The Radbot's traversal of the search space follows patterns found in cell crawling, and it manages to completely surround / encircle a light source. In future work, the repair of the protected light source should be an added priority.

The algorithm itself somewhat resembles the WPA. The Kilobots acting as adhesion sites are similar to the lead wolves in the WPA, calling the rest of the Kilobots to their location. The Kilobots being called, as they traverse the search space, also have the opportunity to become the new adhesion site, just like the scouting and ferocious wolves in the WPA.

It is interesting to see that the hunting patterns of wolf packs can be roughly translated down to the cellular level.

The algorithm might offer merit as a swarm optimization technique of its own. The Radbot swarm is, because of its traversal technique, resilient to being trapped in local optima. Unlike the Kilobots independently moving towards light, the Radbot can scout out in new directions as its component Kilobots move along the Radbot's edge.

Finally, with sustained interest in the field of nanotechnology, this algorithm could prove useful for augmented response for an Astronaut's body when exposed to high-energy radiation in space. If Kilobot-like robots could be made small enough, then perhaps some refined version of this algorithm might be used to introduce Cell Rover like artificial bio-nanocells into the bloodstream of a human in general to boost the capability of the immune system to become more resilient to pandemics like the COVID-19 or 2020.

References

[1] Revathi Ananthakrishnan and Allen Ehrlicher. The forces behind cell movement. *International journal of biological sciences*, 3(5):303, 2007.

[2] Evan Ackerman. A Thousand Kilobots Self-Assemble Into Complex Shapes. IEEE Spectrum, August 2014.

[3] A.P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2005.

[4] Nigel Goddard. Lecture notes on gradient descent. University of Edinburgh, November 2017.

[5] Self-Organizing Systems Research Group. Kilobots labs. Harvard University.

[6] Fredrik Jansson, Matthew Hartley, Martin Hinsch, Ivica Slavkov, Noemí Carranza, Tjelvar SG Olsson, Roland M Dries, Johanna H Grönqvist, Athanasius FM Marée, James Sharpe, et al. Kilombo: a kilobot simulator to enable effective research in swarm robotics. *arXiv preprint arXiv:1511.04285*, 2015.

[7] Ben Joffe. Function 3d examples, 2018.

[8] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, August 1987.

[9] Erol Sahin, Thomas H. Labella, Vito Trianni, J.-L. Deneubourg, Philip Rasse, Dario Floreano, Luca Gambardella, Francesco Mondada, Stefano Nolfi, and Marco Dorigo. SWARM-BOT: Pattern formation in a swarm of self-assembling mobile robots. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, volume 4, pages 6–pp. IEEE, 2002.

[10] N E Shlyakhov, I V Vatamaniuk, and A L Ronzhin. Survey of Methods and Algorithms of Robot Swarm Aggregation. *Journal of Physics: Conference Series*, 803:012146, Jan- uary 2017.

[11] Mamta Patel Nagaraja. [1] Cell Rovers: A Voyage of the Nano-Surgeons, Office of Biological and Physical Research, NASA, 2002.

[12] Hu-Sheng Wu and Feng-Ming Zhang. Wolf Pack Algorithm for Unconstrained Global Optimization. *Mathematical Problems in Engineering*, 2014:1–17, 2014.