2008 IEEE Swarm Intelligence Symposium
St. Louis MO USA, September 21-23, 2008

10.1109/SIS.2008.4668295

Slime Mold as a Model for Numerical Optimization

David R. Monismith Jr. and Blayne E. Mayfield

Abstract— **This work presents a novel approach to single objective optimization using the amoeba Dictyostelium discoideum (Dd), sometimes known as Slime Mold, as its basis. A short explanation of the biological background of Dd is presented. Inspirations taken from existing computational biology and educational simulation studies of Dd also are provided. Based upon previous works, an algorithm for optimization is constructed called the Slime Mold Optimization Algorithm.**

I. Introduction

OVER the past 50 years, many algorithms were introduced to solve particularly difficult optimization problems. Traces of nature are found throughout a number of these algorithms. Creativity and determination have influenced the design of works such as Genetic Algorithms, Ant Colony Systems, and Particle Swarms. The impetus of such algorithms was to solve problems that were and still are considered quite difficult if not intractable.

Based upon another element of nature, this paper attempts to conceptualize the lifecycle of slime mold – a natural formation of amoebae – as a viable optimization algorithm. The product of this work will be a system that solves single-objective optimization problems to near-optimal values within a reasonable amount of time. To solve these types of problems, the lifecycle of one of the more popular slime molds, *Dictyostelium discoideum* (Dd), was studied.

Most efforts at exploring the lifecycle of Dd outside of computational biology have been aimed at educational applications. Such implementations are examples of the lifecycle or show a portion of the lifecycle as an example of the usefulness of parallel thinking [1]-[2]. The lifecycle of Dd also implies some interesting search strategies for optimization [3]. These include directed local random search for vegetative amoebae, dense searching of areas of importance for mounds, directed dense random searching for slugs, and cell death or restarting of the search for many elements of fruiting bodies. All such elements, if put together correctly lead to a new paradigm for hard optimization problems.

To show the value of the Dd lifecycle as an optimization algorithm, the Slime Mold Optimization Algorithm is presented. In doing so, a discussion of the lifecycle of Dd is provided. Leading up to the algorithm, background on its components including the ε-Approximate Nearest Neighbor (ε-ANN) Algorithm, Cellular Automata, and educational uses of Dd are provided. Thereafter, the algorithm is explained in detail with pseudocode, and results for its application to several numerical optimization problems are provided. The results of this paper indicate that slime molds provide a viable alternative to existing optimization algorithms.

Manuscript received May 28, 2008.

D. R. Monismith Jr. is a Computer Science Ph.D. student at Oklahoma State University, Stillwater, OK 74078 USA (phone: 405-744-5668; fax: 405-744-9097; email: monismi@cs.okstate.edu).

B. E. Mayfield, is an Associate Professor with the Computer Science Department at Oklahoma State University, Stillwater, OK 74078 USA. (email: bem@cs.okstate.edu).

II. Biological Background

The cells of a slime mold are known to biologists as the amoeba *Dictyostelium discoideum* (Dd) and a number of other genuses [4]. Dd cells progress through number of lifecycles, lending themselves to be easily modeled as cellular automata (CA) [1]-[9]. Figure 1 illustrates the lifecycle of Dd succinctly; later this lifecycle will be adapted to solve optimization problems. Dd cells move about independently until starvation [4]-[9]. During starvation, Dd emit a chemical known as cyclic Adenosine Monophosphate (cAMP). Cells have several options at the point of starvation. They may form microcysts or macrocysts, which go dormant underground until some factor (such as a climate change) revives them [4]. The last form that Dd undergo during starvation is a period of aggregation or grouping together. Thereafter, as shown in Figure 1, cells group together in streams and eventually form a mound [4]-[9].

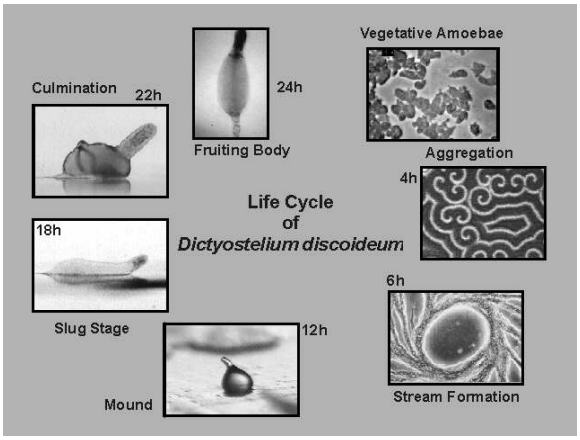

Figure 1. Lifecycle of Dictyostelium discoideum [10]. Used with permission from Florian Siegert, © 1998.

The mound forms a slime sheath about itself to protect from predatory multicellular organisms such as nematodes (small worms) [4]. Once the mound is complete, cells orient themselves to form a head and a tail [4]-[9]. Note that at this point, the slug (commonly known as a slime mold) is not a multicellular organism. It is a group of single celled organisms working toward a common goal.

A. Single Celled Movement

When living as single cells, Dd eat decaying logs, humus, and bacteria [4]. These cells move about through soil or within decaying material. Dd have long arms called pseudopods that contain sensors [4]-[9]. Imagine if you had a nose at the end of your arm in addition to the nose on your face. You could judge the direction and strength of scents by extending your arm in multiple directions. Dd search for food in this fashion [11]. If there is no food available, Dd wander about aimlessly searching for food. Once Dd acquire a significant amount of resources, they may procreate [4]. Procreation occurs via binary fission, that is, Dd multiply by dividing into two Dd cells after consuming enough food [4].

B. Aggregation

With no food source available, Dd wander about aimlessly. But, with enough Dd in one area, a group of cells can alleviate this problem. After 6-10 hours of starvation, Dd cells become chemotactically sensitive to cAMP, and most cells are able to send and receive cAMP signals [11]. Once these cells exhibit this capability, they begin to sense and move toward a cAMP source. Starving cells also leave cAMP in a trail in the same way ants drop pheromones [4]-[9], [11], [12]. Aggregation occurs when a starving cell begins to emit the cAMP signal. This cell is known as a pacemaker cell [4]-[9]. Once the pacemaker sends out a signal, other nearby Dd cells are attracted to the emitted cAMP signal. These cells also leave a trail of cAMP that induces starving cells that are further away to move toward the pacemaker [4]-[9]. While aggregating toward the pacemaker, cells form streams or pathways for other cells to follow [4]-[9]. These are localized groups of cells, allowing Dd cells to lead each other to the pacemaker by their emissions of cAMP [4]-[9].

C. The Mound

After aggregating, cells join together about the pacemaker cell and enter an intermediary stage known as a mound. The mound is a precursor to a mobile slug of multiple amoebae working together. The mound consists of hundreds or thousands of amoebae working together towards a common goal – survival of the group in the face of starvation. Amoebae within the mound organize themselves to form both a head and a tail [4]-[9]. At this time, it is also determined which cells shall be prestalk (those cells forming the head) and which shall be prespore (those in the tail) cells when the fruiting body is formed [4]-[9]. Evidence exists to show that cells that led a better life (had more food reserves and better living conditions) are preferred to become prespore rather than prestalk cells [4]. Nonetheless, those cells that just divided (reproduced) have a lower tendency to become prespore than cells that have ample food reserves and have not divided because they have fewer food reserves. Therefore, fit cells are preferred to become prespore over those that are unfit.

The mound stage also includes formation of a slime sheath. Formation of the slime sheath begins during aggregation, when each cell begins to form a portion of this sheath. The sheath allows for conservation of volume of the slug and serves to protect it from predators such as nematodes [4]. The sheath also to aids in formation of the fruiting body's stalk, and prevents predators from entering and amoeba from exiting the slug.

D. The Slug

The slug (or slime mold) is a mobile self-contained group of Dd. Slime molds have both phototactic (light sensitive) and chemotactic (chemically sensitive) properties, enabling them to migrate to a suitable area for spore production [6]-[9]. Migration occurs as a result of the movement of the amoebae contained within the slug. Migration may take hours or days, and the slug may continue migrating until the amoebae either find a point of culmination or expire [4]. Once the amoebae have formed a slug, the goal of the slug is to move to a position where a fruiting body might be formed so Dd spores may be dispersed by wind to another area to ensure survival of the group [4]. Amoebae within the slug are divided into two sections – a head and a tail. Cells in the head are prestalk amoebae that are both chemotactically and phototactically sensitive [4]-[9]. They also send signals to the tail of the slug. Such signals direct the cells in the tail to move opposite the gradient of light and along of the gradient of noxious chemicals like ammonia [5]. By emitting cAMP, amoebae in the tail, which are mainly prespore cells, move forward in a column-like fashion [4]. This emission of cAMP causes cells in the tail to propel the slug like treads on a tank. This process continues until the slug reaches a location with a significant light source or until resources have been depleted. If all resources have been depleted, the amoebae die; however, if a light source is available, culmination occurs and a fruiting body is formed [4]-[9].

E. The Fruiting Body and Dispersal

During culmination and formation of the fruiting body, prestalk amoebae form a stalk. Each prestalk cell dies as it becomes part of the stalk [4]-[5]. As the stalk is formed, prespore cells crawl up the stalk to form a head consisting of spores, similar to a dandelion. Once the spores on the fruiting body have completely formed and reached the top of the stalk, it is up to birds, invertebrates, and the wind to disperse the spores [4]. When spores arrive at another location, they may lie dormant for some time. After a period of dormancy, spores form new amoebae, and the lifecycle of Dd then begins again [4].

III. Algorithmic Background

Before discussing the slime mold optimization algorithm, it is necessary to review several algorithmic background topics. Research in computational biology provides many methods to model slime mold, including statistical models, partial differential equations, cellular automata (CA), and combinations thereof [1]-[9], [11], [13]. In creating an algorithm, the authors made use of cellular automata to represent Dd. Therefore, a review of cellular automata is provided with formalisms. Moreover, formalism of a simple model of Dd as a 2D cellular automaton in a grid-based state space is provided. Thereafter, a short discussion of the approximate nearest neighbor (ε-ANN) algorithm is

provided, as a mesh based on ε-ANN will be used for the state space of the slime mold optimization algorithm in place of a grid.

A. Cellular Automata

The state space L of a cellular automaton may be 1, 2… or N–dimensional. This space may have boundaries or it may be infinite [14]. The boundary conditions and/or shape of the state space will be denoted as the set B. Each location in the state space will be referred to as a cell. Each cell within the state space may take on a particular set of values. This set of values will be referred to as Σ with discrete values ranging from zero to some maximum value k [14].

$$\Sigma \equiv \{0,1,...,k\} \tag{1}$$

Each cell assumes a position within the state space and is assumed to take on a value from Σ at time t [14]. Positions within the state space are represented as discrete points in space. These points may be represented as a tuple including the location of the point along each dimension, for example (i, j, k …). In general, the size of this tuple is the number of dimensions of the state space. Therefore, we may define the value σ of a particular cell at a given time t as follows [14]:

$$\sigma_{i,j,k,\ldots}(t) \in \Sigma \tag{2}$$

Where *(i, j, k ...)* represents the position of the cell, and t is the time at which this value exists [14].

Rules are also necessary for specification of updates to the state space. We may define a rule φ as a mapping from the values of zero or more existing cells within the state space (quite possibly the neighborhood of the cell in question) to one value. Each rule may be defined via the following mapping [14]:

$$\varphi : \Sigma \times \Sigma \times \ldots \times \Sigma \to \Sigma \tag{3}$$

Therefore, rules are functions that allow update of the state space to a new state based upon values in a previous state. Note that updates are often made to the neighborhood of a given cell, but neighborhoods may be defined in many ways. For example, a von Neumann neighborhood is defined as the immediate nearest neighbors of a given cell within a given radius, whereas the Moore neighborhood of a cell is defined as all cells within a distance of $x \cdot \sqrt{n}$ of a given cell where the cell is in an n dimensional space, and x is the radius of the neighborhood [15]. The figure 2 illustrates examples of von Neumann and Moore Neighborhoods for 2 dimensional cellular automata.

Note that a neighborhood N may be represented as simply the nearest neighbors, a group of neighbors, or in other forms than those shown above. For example, a particular cell could have a countably or uncountably infinite number of neighbors [14].

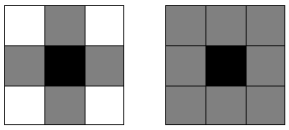

Figure 2. von Neumann (left) and Moore Neighborhoods (right) for Discrete Cellular Automata [15].

A cellular automaton A is defined as follows:

$$A = (L, N, \Sigma, \Phi, B) \tag{4}$$

Where L represents the state space; N represents the neighborhood; Σ represents the starting state; Φ represents the set of rules; and B represents the set of boundary conditions [14]. If necessary, additional terms may be added to the formalism to encompass more complex automata.

B. Representation of Dd as a CA

Dd has been formalized as a cellular automaton by many sources. In most cases this formalism is provided in two or three dimensions. In 2D, the state space for Dd is most easily viewed as a grid, and is typically implemented as a torus. Dd populate this grid and make modifications to it. The grid (i.e. the cellular automaton) is updated based upon discrete time steps. A time step includes allowing one movement for each amoeba located within the state space. The formalism below is based upon both educational and biological references. It models each amoeba in the slime mold as an individual and limits the amount of cAMP each amoeba can deposit to a discrete value. The grid or world (i.e. the state space) depends upon each existing neighborhood (simulated as a Moore neighborhood) and the amoebae that update these neighborhoods. Several rules exist for updating the world: (1) Dd deposit a certain amount of pheromone within the cell they currently occupy, (2) some of the pheromone within each cell evaporates after each time step, and (3) pheromone spreads from one cell to another via a smoothing function [1]-[2].

Rules for the slime mold world in [1] and [2] are quite simple. Evaporation removes one unit of cAMP from each cell in the world as long as cAMP exists.

$$\sigma_{i,j}(t+1) = \begin{cases} \sigma_{i,j}(t) - 1, \sigma_{i,j}(t) > 0 \\ 0, \sigma_{i,j}(t) = 0 \end{cases} \tag{5}$$

For neighborhood update of cAMP within the world, a smoothing function that operates over a Moore neighborhood of size one is used [1].

$$\sigma_{i,j}(t+1) = \begin{cases} \frac{1}{9} \sum_{q} \sum_{r} \sigma_{q,r}(t), \sigma_{i,j}(t+1) < k \\ k, \sigma_{i,j}(t+1) \geq k \end{cases} \tag{6}$$
$$q \in \{i-1, i, i+1\}$$
$$r \in \{j-1, j, j+1\}$$

cAMP is also updated when an amoeba occupies a particular cell within the world. If a cell was at position (i, j) within the grid at time t, it will leave a certain amount of

cAMP in that cell once it moves to another position at time t+1. An arbitrary value may be chosen for the amount of cAMP to leave behind. For example, [1] chose to use a value of eight for every amoeba dropping cAMP. The update formula may then be expressed as follows below [1].

$$\sigma_{i,j}(t+1) = \begin{cases} \sigma_{i,j}(t) + x, \sigma_{i,j}(t) < k \\ k, \sigma_{i,j}(t) \geq k \end{cases} \quad (7)$$

Where x is the amount of cAMP dropped by the amoeba that was in cell (i, j) at time t, and k is the maximum amount of cAMP that may be deposited within a cell on the grid.

Each amoeba is also assumed to follow a given set of rules. In the simulation provided by [1] and [2], all amoebae are assumed to be starving. All starving amoebae are assumed to deposit cAMP. These amoebae are attracted to either a random, unoccupied location if all neighboring cells contain the same amount of cAMP or the neighboring location with the greatest amount of cAMP. If the neighboring location that was chosen is occupied, the amoeba does not move. Otherwise, the amoeba drops its cAMP and moves to the neighboring cell.

These sets of rules and automata including the slime mold world and the amoeba that exist within it may be formalized as follows:

$$World = (L, N, \Sigma, \Phi, B, A, T, S) \quad (8)$$

Where L is the state space of the world, N is the neighborhood for each cell, Σ is the set of local values (cAMP value) for each cell, Φ is the set of rules for the world, B is the set of boundaries for the world, A is the set of amoebae existing within the world, T is the set of transition rules for the amoebae, and S is the starting state of the world and the amoebae within it.

C. ε-Approximate Nearest Neighbor

The use of a grid to represent the state space of the Slime Mold is appropriate for educational and biological purposes, but it is not practical for optimization. With d dimensions in a space and assuming a square space, the use of a grid will cost $O(m^d)$ memory, m being the number of cells along one dimension of the grid. Such memory usage would not be acceptable for even a small number of cells along one dimension with a high dimensional space. For example, assuming m = 10 and assuming a 32 dimensional space, the number of cells within a grid would be 10^{32}. A grid is not a viable data structure for use in an optimization algorithm that models the lifecycle of Dd.

A sparse mesh is much more suitable for optimization than a grid. Using a naïve nearest neighbor algorithm, a mesh that encompasses all of the starting points of the slime mold optimization algorithm may be formed. The use of such a mesh will allow for considerable savings in memory usage. Memory usage for the mesh would be O(kdn) where k is the number of neighbors being considered, d is the dimension of the space, and n is the number of starting points. Unfortunately, the naïve nearest neighbor algorithm takes $O(n^3)$ to run to completion [16]-[17]. To escape this problem, the ε-Approximate Nearest Neighbor (ε-ANN), an optimal nearest neighbor algorithm created by [16] and [17], may provide an alternative to the traditional brute force nearest neighbor algorithm that takes $O(n^3)$. It also reduces the amount of time required for the nearest neighbor algorithm to O(dn log n) and reduces query times for neighbors to O(d log n) [16]. [17] provides the ε-ANN library, which is used to implement the mesh described above.

IV. Methods

The goal of this research was to create a model of the slime mold that would solve single objective optimization problems within the continuous domain by making use of the lifecycle of Dictyostelium discoideum (Dd). This lifecycle is built of several independent but interesting sections. Amoebae in the optimization algorithm are assumed to have the following traits: a state of vegetative, aggregative, mound, slug, or dispersal; a current state space location; a current objective function value; a personal best state space location; a personal best objective function value; and the amount of time since the last personal best improvement [18]-[19].

A. Vegetative Amoebae

Biological amoebae perform a random search for food within the soil. They move using their pseudopodia as sensors to detect nearby food sources. These pseudopodia are not always completely accurate, but they work, for the most part, to direct the amoebae toward food when it is available [4]. A computational representation of the vegetative amoebae should thus perform a semi-random search within the state space. To perform this search, amoebae are given the ability to "extend their pseudopodia" in multiple directions to perform a local search. This local search could be represented by any strategy, but that is beyond the scope of this paper. For the purposes of this paper only one strategy will be considered.

As mentioned before, the search strategy allows simulated amoebae to extend pseudopodia in multiple directions. The number of pseudopodia and the length of their search radius may vary based on the problem in question. The general search strategy for each pseudopod is as follows: the search distance is proportional to the average distance from each neighboring amoeba. The search location is determined using the following formula:

$$x_i = c \cdot rand(0,1) \cdot E(d_{neighbor}), \quad (9)$$

Where c is a constant less than one, x_i is the location of the i^{th} amoeba, and $E(d_{neighbor})$ is the average distance to a neighboring ε-ANN mesh point as described in algorithm 4.6. With a probability of p, the amoeba will move toward one of the pseudopodia chosen at random. With a probability of 1-p, the amoebae will perform a roulette wheel choice of a pseudopod weighted in favor of the pseudopod with the best objective function value [20]. It is also necessary to

ensure that the amoeba keeps track of its closest lattice point within the state space to ensure proper update of the state space upon starvation. The pseudocode for the vegetative amoeba stage is provided in algorithm 4.1.

Algorithm 4.1: Vegetative Movement

1. Using (9), the amoeba searches in k directions with its pseudopods, evaluating the objective function at each position.
2. Movement is based on a roulette wheel with better pseudopods having a higher chance of being chosen.
3. If amoeba is starving, based on (10), change the state of the amoeba to AGGREGATIVE.

B. Aggregation and Stream Formation

Simulation of starvation is necessary to initiate an aggregative state for amoebae. Therefore, it is assumed that during the vegetative state, amoebae are "fed" by values from the solution space. So, amoebae must have a memory of their personal best (minimum objective value and location attained) [18]-[19]. Updating this personal best may count as one feeding. If an amoeba becomes trapped within a local optimum, it will fail to update its personal best. The following formula illustrates one method to determine if an amoeba is starving [20]:

$$p\left(x < \frac{\mathrm{t_{unimproved}}}{\mathrm{t_{lifetime}}}\right) \qquad (10)$$
$$g(x) = 1, \text{ for } 0 \le x \le 1,$$

Where x is a random variable with probability density function g, $\mathrm{t_{unimproved}}$ is the amount of time that the amoeba has gone without improving its personal best, and $\mathrm{t_{lifetime}}$ is the number of time steps that the amoeba has existed since its last dispersal.

Once a cell is determined to be starving, it emits cAMP. Thus, each aggregating cell may be represented as part of a cellular automaton. In doing so, the world is represented as a mesh formed from the ε-ANN algorithm instead of a grid or multidimensional array [1]-[2], [13], [20]. Each point in the tree may assume a discrete set of values in the range {0, 1,...,k}. k is assumed to be reasonably large (> 100) for (11), (12), and (13) to apply. Moreover, the amount of cAMP deposited at mesh location $\mathrm{x_{ANN}}$ at time t+1 is dependent upon the amount of cAMP deposited at position $\mathrm{x_{ANN}}$ at time t in addition to an amount of cAMP present at that position by the amoeba that was at that position at time t (if any). The mesh used in this research for the locations where cAMP may be deposited is shown in Figure 3.

Note that an amoeba deposits an amount of cAMP at its current location proportional to the norm of its current objective value with respect to the global (overall) best objective function value. The following formula exhibits the cAMP deposit rule [1], [20].

$$\sigma_i(t+1) = \begin{cases} \sigma_i(t) + c\left(norm\left(\dfrac{f(x_{current})}{f(x_{pBest})}\right)\right), \sigma_i(t+1) < \mathrm{k} \\ \mathrm{k},\ \sigma_i(t+1) \ge \mathrm{k}. \end{cases} \qquad (11)$$

Rules within this cellular automaton also allow for smoothing (neighborhood update) and evaporation after all updates to all positions have been made. The evaporation rule consists of removal of a single unit of cAMP from each node (lattice point) after all updates have been made for a single time step. Evaporation may be expressed through the following rule [1]:

$$\sigma_i(t+1) = \begin{cases} \sigma_i(t) - 1, \sigma_i(t) > 0 \\ 0, \qquad \sigma_i(t) = 0. \end{cases} \qquad (12)$$

The neighborhood update or smoothing rule may be expressed as a convolution operator; that is, an operator that works on the entire box decomposition tree at once to perform an update involving each surrounding node or nearest neighbor. This operation may be expressed through the following rule [1]:

$$\sigma_q(t+1) = \begin{cases} \dfrac{1}{N+1}\sum_i \sigma_i(t), \sigma_q(t+1) < k \\ k, \sigma_q(t+1) \ge k. \end{cases} \qquad (13)$$
$$q \in \{1,\ldots,N\}$$
$$i \in \{\text{The Neighbors of } \sigma_q(t) \text{ and } \sigma_q(t) \text{ itself}\}.$$

Using each of the three rules above in conjunction will allow starving cells to congregate about a single point. These rules will also attract other starving cells to this centralized point. The pseudocode for the aggregation stage is provided in Algorithm 4.2.

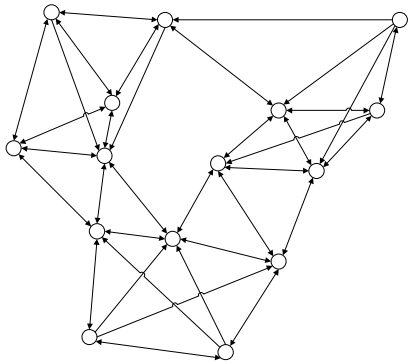

Figure 3. Approximate Nearest Neighbor Mesh, 2-D with 4 neighbors per node.

Movement in the aggregation stage is achieved by a similar formula to that of particle swarm optimization. Using this formula, an amoeba moves in three different ways. It moves in the direction of the strongest nearby cAMP source, in the direction of its personal best, and it also makes a random movement scaled by the average distance to its nearest neighbors. The formula for movement in the aggregation stage is listed below.

$$v_i(t+1) = c_1 v_i(t) + c_2(x_{bestANNNeighbor}(t) - x(t)) + c_3(x_{pacemaker}(t) - x(t)) + c_4 E(d_{neighbor}) N(0,1)$$
$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (14)$$

In (14), x_i is the location of the amoeba, v_i is the velocity of the amoeba, $x_{bestANNNeighbor}$ is the nearest mesh location with the largest amount of cAMP (excluding the amoeba's current mesh location), and $x_{pacemaker}$ is the location of the pacemaker amoeba (i.e. the aggregating cell with the best objective value). The constants c_1, c_2, c_3, and c_4 are heuristic constants similar to those in [19]. $E(d_{neighbor})$ is the mean distance to the amoeba's nearest neighbor within the slug's mesh.

Algorithm 4.2: Aggregation

1. If a pacemaker amoeba exists, move via (14); otherwise, perform vegetative movement.
2. Drop cAMP at closest mesh point using (11).
3. If no pacemaker exists, this amoeba becomes the pacemaker and emits cAMP from its location.
4. If formula (15) is satisfied, at the next time step, convert all amoebae at the pacemaker's location and within a radius r to the MOUND state at the next timestep.

C. Mound Formation

A mound is formed based on probability alone. As the number of starving amoebae in one distinct lattice point increases and surpasses some preset threshold, the probability of forming a mound also increases. Values below this threshold are not considered to have the ability to form a mound. One possible formula for determining if a mound should be formed follows below [20]:

$$p\left(x > \frac{A_i - A_{Min}}{A_{Max}}\right) \quad (15)$$
$$g(x) = 1, \text{for} 0 \le x \le 1.$$

Where x is a random variable, g is a pdf for x, A_i is the number of amoebae at position i in the grid, A_{Min} is minimum threshold needed to form a mound, and A_{Max} is the maximum threshold needed to form a mound. Once a mound is determined to be at a particular location, a dense lattice is formed over the points located within the mound using each of their locations. The approximate nearest neighbor algorithm provided by [16] and [17] is utilized here.

Algorithm 4.3: Mound Formation

1. Add amoeba to it's pacemaker's ANN mesh based on its current location.
2. At end of this timestep, create entire mesh for this mound with locations for cAMP.
3. Convert all amoebae within the mesh to the SLUG state.

This small step represents the formation of the mound. Once this formation occurs, we allow for local search over the area of the mound which is the initialization of the slug stage. Pseudocode for the mound stage is provided in Algorithm 4.3

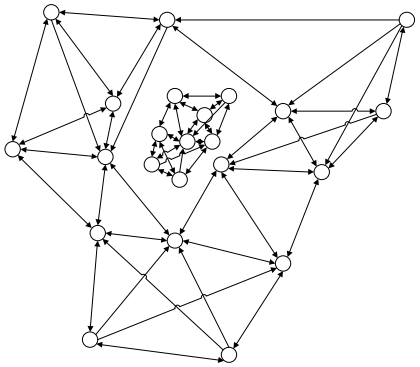

Figure 4. Approximate Nearest Neighbor Mesh with Mound, 2-D with 4 neighbors per node.

D. Slug Movement

During the slug stage, updates are performed using the rules from the sparse lattice on the dense lattice for the slug. Update is, however, only performed on the locations within the denser lattice. If necessary, this denser lattice is updated to better reflect the location of the slug. A dense lattice reset is performed after the centroid of the amoebae locations surpasses a threshold distance from the original centroid of the slug. The following formula is an example of how this property may be enforced [20].

$$p\left(x < \frac{dist(currentCentroid, oldCentroid)}{std(oldLatticePoints)}\right) \quad (16)$$
$$f(x) = 1, \text{for} 0 \le x \le 1$$

Pseudocode for the slug stage is provided in Algorithm 4.4.

Algorithm 4.4: Slug Movement

1. Perform local search for this amoeba via (17).
2. Update the function value, and personal best if necessary.
3. If, after performing all movements for this mound, (16) is satisfied, reset the mesh for the slug.
4. If (18) is satisfied, convert this amoeba to DISPERSAL state.

Movement for each amoeba within the slug also is performed using a formula similar to that of particle swarm optimization [18]-[19]. This formula is provided below.

$$v_i(t+1) = c_1 v_i(t) + c_2(x_{bestNeighbor}(t) - x(t)) + c_3(x_{slugBest}(t) - x(t)) + c_4 E(d_{neighbor}) N(0,1)$$
$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (17)$$

Where x_i is the location of the amoeba, v_i is the velocity of the amoeba, $x_{bestNeighbor}$ is the location of the amoeba's nearest neighbor with the best personal best, and $x_{moundBest}$ is the location of the best value found within the slug. As in (14), here c_1, c_2, c_3, and c_4 also are heuristic constants similar to those in [19]. $E(d_{neighbor})$ is the mean distance to the amoeba's nearest neighbor within the slug's mesh.

E. Culmination, Fruiting Body, and Dispersal

Representation of culmination, the fruiting body, and dispersal in this algorithm is quite simple. Once there are too few personal best updates to the slug, it must be dispersed to continue finding values that are closer to the optimum than the current global best value. To do so, the following formula is used to determine when dispersal will occur [20]:

$$p(x > \frac{\text{Number of Updates}}{\text{Update Threshold}}) \qquad (18)$$
$$f(x) = 1, \text{ for } 0 \le x \le 1$$

Based upon the formula above, dispersal will occur with higher frequency when there are fewer updates to personal bests of amoebae within the slug. Therefore, amoebae within the slug must keep a memory of their personal bests if the algorithm is to be optimal. So, immediately before random restart, each amoeba is assumed to be in the culmination and fruiting body stage. When random restart occurs, amoebae are assumed to disperse. Pseudocode for dispersal is provided in Algorithm 4.5.

Algorithm 4.5: Dispersal

1. Chose a random location within the state space for this amoeba.
2. Place the amoeba at that location.
3. Evaluate the objective function at that location.
4. Convert this amoeba to vegetative state.

F. Slime Mold Optimization Algorithm

As indicated in the previous sections, the Slime Mold Optimization Algorithm (SMOA) is a multi-step iterative algorithm that mimics the lifecycle of Dd to a large extent. Amoebae within the algorithm may take on a number of states: vegetative amoeba, aggregating amoeba, mound, slug, fruiting body, and dispersal. All amoebae are assigned uniform random locations and begin the algorithm in the vegetative state and are given time to search for local optima (i.e. food). Based on their initial locations, a mesh is formed using the ANN library. Each amoeba is given the ability to search based on a simple stochastic local search algorithm as described in section 4.1. Pseudocode for SMOA is provided in algorithm 4.6.

V. Results

The slime mold optimization algorithm was tested on a group of 5 functions and each result was averaged over 100 simulations. The slime mold algorithm was allowed to run for 1000 iterations on each of the functions listed in Table 1. These functions were chosen to show the usefulness and difficulties of the algorithm with respect to some existing optimization problems with known solutions and with respect to some that are known to cause problems with other algorithms. In particular, several of the functions were chosen because of the problems they cause with some existing optimization algorithms. The mineshaft function chosen includes two mineshafts on a plane and it was created by the author. The inclusion of these very narrow mineshafts makes solving this function particularly difficult.

Algorithm 4.6: Slime Mold Optimization Algorithm

1. For each amoeba
2. Begin amoeba at a uniformly distributed random location within the search space.
3. Evaluate the objective function at the current location.
4. Store the objective function value and location as the amoeba's current location, current objective function value, personal best objective function value, and personal best location.
5. Set the amoeba's state to VEGETATIVE.
6. End for
7. Archive the best objective function value from the amoebae.
8. Input the locations of each amoeba to ε-ANN.
9. Create a mesh based on the results of ε-ANN.
10. For each amoeba
11. Switch (Amoeba state)
12. Case VEGETATIVE: VegetativeMovement; End Case
13. Case AGGREGATIVE: Aggregation; End Case
14. Case MOUND: MoundFormation; End Case
15. Case SLUG: SlugMovement; End Case
16. Case DISPERSAL: Dispersal; End Case
17. End Switch
18. End for

VI. Conclusion

In this work, implementation of a novel AI inspired optimization algorithm was presented. The slime mold algorithm performs well when searching for optima where mineshafts exist and in high dimensional problems. Future work will test other types of problems, and compare the results of the slime mold algorithm to other optimization algorithms. Other possibilities for future research include modifying the local search algorithm that is used in the vegetative amoeba stage and the methodology used for the slug stage.

Acknowledgment

The authors thank John P. Chandler, Ph.D. and Andrea Monismith, DVM for their assistance with this paper.

References

[1] Matthews, J. "Slime Mold Simulation – Java Applet", URL: http://www.generation5.org/content/2002/slimejava.asp.

[2] Resnick, M. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*, Massachusetts Institute of Technology Press, 1994.

[3] Rothermich, J. A. *From Multicellularity to Cell Based Optimization*, Master's Thesis, The University of Birmingham, September 2002.

[4] Kessin, R. H. *Dictyostelium: Evolution, Cell Biology, and the Development of Multicellularity*, Cambridge University Press, Cambridge, UK, 2001.

[5] Marée, A. F. M. and Hogeweg, P. "How amoeboids self-organize into a fruiting body: Multicellular coordination in *Dictyostelium discoideum*", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 98, pp. 3879-3883, March 27, 2001.

[6] Marée, A. F. M., Panfilov, A. V., and Hogeweg, P. "Phototaxis during the slug stage of *Dictyostelium discoideum*: a model study",

Proceedings of the Royal Society of London B, Vol. 266, pp. 1351-1360, 1999.

[7] Marée, A. F. M., Panfilov, A. V., and Hogeweg, P. "Migration and Thermotaxis of *Dictostelium discoideum* Slugs, a model study", *Journal of Theoretical Biology*, Vol. 199, pp. 297-309, 1999.

[8] Erban, R. and Othmer, H. G. Taxis Equations for Amoeboid Cells, Submitted to the *Journal of Mathematical Biology*, 2006.

[9] Savill, N. J. and Hogeweg, J. Modeling Morphogenesis: From Single Cells to Crawling Slugs, *Journal of Theoretical Biology*, Vol. 184, pp. 229-235, 1997.

[10] Siegert, F. Dictyostelium Homepage ZI-Munich, URL: http://www.zi.biologie.uni-muenchen.de/zoologie/dicty/dicty.html.

[11] Dallon, J. C. and Othmer, H. G. "A discrete cell model with adaptive signaling for aggregation of *Dictyostelium discoideum*", *Philosophical Transactions of the Royal Society of London B*, Vol. 352, pp. 391-417, 1997.

[12] Dorigo, M. and Stützle, T. *Ant Colony Optimization*, Massachusetts Institute of Technology Press, Cambridge, MS, 2004.

[13] Segel, L. A. "Computing an organism", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 98, pp. 3639-3640, March 27, 2001.

[14] Ilachinski, A. *Cellular Automata: A Discrete Universe*. World Scientific Publishing Co., River Edge, NJ, 2001.

[15] Auer, K. and Norris, T. ""ArrierosAlife" a Multi-Agent Approach Simulating the Evolution of a Social System: Modeling the Emergence of Social Networks with "Ascape"", *Journal of Artificial Societies and Social Simulation*, Vol. 4, No. 1, January 31, 2001.

[16] Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. "An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions". *Journal of the ACM*, Vol. 45, No. 6, pp. 891-923, Nov. 1998.

[17] Mount, D. M. and Arya, S. "ANN: A Library for Approximate Nearest Neighbor Searching", University of Maryland, URL: http://www.cs.umd.edu/~mount/ANN/.

[18] Coello Coello, C. A. and Lechuga, M. S. "MOPSO: a proposal for multiple objective particle swarm optimization". *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002, Vol. 2, pp. 1051 – 1056, 12-17 May 2002.

[19] Kennedy, J. and Eberhart, R. "Particle Swarm Optimization," *Proc. of the IEEE Int. Conf. on Neural Networks*, Vol. 4, pp. 1942–1948, Nov. 1995.

[20] Yen, G. G. and Lu, H. "Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation" *IEEE Transactions on Evolutionary Computation*, Vol. 7, pp. 253 – 274, June 2003.

[21] Chen, D. *Comparisons Among Stochastic Optimization Algorithms*. Master's Thesis, Oklahoma State University, OK, 1997

[22] GEATbx: Examples of Objective Functions, URL: http://agat.ely.pg.gda.pl/~mkwies/dyd/sro/geadocu/fcnindex.html.

TABLE 1: TEST FUNCTIONS AND RESULTS [21]-[22].

Functions	Formulas	Average (f_a) and Best (f_b) Results	Domain	f(x*) = f*
Rosenbrock's Function	$f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$	f_a = 2.07e-10 f_b = 9.77e-14	$x_1, x_2 \in [-2,2]$	$f(1,1) = 0$
Mineshaft Function	$f(x) = -5 \cdot \exp(-1000 \cdot (x_1 - 0.5)^2 - 1000 \cdot (x_2 - 0.3)^2) - 7 \cdot \exp(-2000 \cdot (x_1 - 0.8)^2 - 2000 \cdot (x_2 - 1.3)^2)$	f_a = -6.96 f_b = -7	$x_1, x_2 \in [-2,2]$	$f(0.8,1.3) = -7$
Osborne's Function 1	$f(x) = \sum_{i=1}^{33} ((x_1 + x_2 \exp(-x_4 t_i) + x_3 \exp(-x_5 t_i)) - y_i)^2$ $t = 10.0 \cdot (i - 1)$ Values for y are provided in [21].	f_a = 0.0228 f_b = 4.25e-4	$x_1, x_2, x_4, x_5 \in [0,3]$ $x_3 \in [-3,0]$	$f(0.3753, 1.9358, -1.4647, 0.01287, 0.02212) = 0.546e-4$ Note that this is an approximate value from [21].
Spherical Contours Function	$f(x) = \sqrt{\sum_{i=1}^{n} x_i^2}$	f_a = 0.00828 f_b = 0.00134	$x_i \in [-10,10]$, $i \in \{1,2,...,32\}$	$f(0,...,0) = 0$
Box & Betts Exponential Quadratic Sum	$f(x) = \sum_{i=1}^{10} g(x)^2$ $g(x) = \exp(-0.1ix_1) - \exp(-0.1ix_2) - (\exp(-0.1i) - \exp(-i))x_3$	f_a = 2.11e-11 f_b = 1.42e-13	$x_1 \in [0.9,1.2]$ $x_2 \in [9,11.2]$ $x_3 \in [0.9,1.2]$	$f(1,10,1) = 0$