Generation and Repair of Organisms Within an Amorphous Computer Using a Virus-Cell Communication Model

To be submitted to the special issue of the
Journal of Autonomous Robots
on Swarm Robotics

Yuriy Brun
6.978 Final Paper
Prof. Abelson, Prof. Sussman, Dr. Nagpal
MIT Laboratory for Computer Science
200 Technology Square
Cambridge, MA 02139 USA
brun@mit.edu

Abstract

It has been demonstrated that an amorphous computer can organize its elements into various shapes as designed by the user. The algorithms for such organizations are very robust to element malfunction during the creation process, but not to malfunction or death afterwards. In this paper I propose a modified model of an amorphous computer that allows the programmer to think as a cell that is being infected by a virus. The processing elements are simple and communicate via executable, virus-like messages. The viruses modify the processors as they see fit to create stable, self-repairing organizations.

Table of Contents

List of Figures

1. Introduction

Animal cells are capable of self-organization into tissues such as bones and skin. In turn, some viruses are capable of destroying such tissues and reorganizing the cells for new tasks. The goal of my work is to use the virus-cell interaction paradigm to design a model for programming a large distributed computer, an amorphous computer, which is composed of thousands of cheap and unreliable processors.

As a demonstration of the power of the virus-cell model, I design and implement two algorithms based on the paradigm: one to create simple creatures, or "line organisms," and the other to repair such organisms if cells happen to die or malfunction. The model assumes the cells, or processors, to be extremely simple and unreliable, thus cheap, making production and application of thousands of such processors to a single problem feasible. The processor cells have some communication power with other cells within a small communication radius. Each cell has some memory and a pseudorandom number generator for tiebreaking. The cells pass around and execute viruses, or small fragments of executable code, which in turn can modify the cells by reading and writing to memory. In a more advanced cell model, the viruses can also prompt the cell to move. A virus holds the power to ask the cell to replicate that virus and also to mutate it into a different virus.

I found the generation of algorithms, or virus programs, to be surprisingly simple. The robustness for cells dead at creation is built into the model and the user only needs to think in terms of "if I were a cell that just got infected, what would I need to do?" Algorithms that deal with cell malfunction and death at a later time are similarly structured. I was able to produce algorithms that generated line organisms and also ones that repaired them. I found the system to be extremely robust to cells that are dead on creation and capable of repairing itself if cells died after the organism was created. That is, the system can always create the line organism as long as there are enough live cells near one another for such an organism to exist. Extra dead cells do not disrupt the

process. The system also dealt fine with cells that were mutated to malfunction by acting randomly. The system was even robust, in some ways, to cells that were designed to thwart its efforts. For example, a cell could recruit the cells around it to form into organisms again and again, using up all the cells. If cells are not allowed to partake in more than one organism, this thwarting would effectively kill cells; however, if any cell can be a part of more than one organism, the saboteur cell slows down the system slightly by making cells compute more but does not change the ability to create and repair other line organisms.

2. Background

Most of the work in the field has concentrated on allowing the cells to be programmable and making the messages passed between the cells simple (Abelson, et al, pp74-82). The key difference in my model is that the cells are assumed to be simple and have no knowledge of the task they will perform. When manufactured, the cells are all programmed with a common operating system, whose job is only to wait for and execute external messages. The messages, or viruses, passed between the cells, are executable and code all the information about the expected cell behavior. Thus by inserting different viruses into the system, the same cells can be used to create line organisms, predict tension failures in buildings, etc. The cells can be manufactured without the knowledge of their final purpose, making the creation process simpler. The only aspect of the cells that needs to be predetermined are the sets of sensors and actuators each cell possesses to perform its type of tasks.

William Butera designed a paintable computer, a 2mm by 2mm square computer that communicates via radio with neighboring computers. Thousands of paintable computers can be thrown into a paint bucket and painted onto a surface, such as a bridge. The computers could then, for example, communicate to calculate the tensions of various parts of the bridge and predict possible structural problems. Butera's computers are universal machines (Butera, pp43-45). My work assumes cells to be able to execute

some simple language and does not assume, although also does not rule out universality. The complexity of the system is removed from the creation of the hardware, the cells, and moved to the software, the virus programs, which can be tested and debugged in simulation before and after the cell manufacture.

Daniel Coore has developed algorithms he called point-growing that are sufficient for creating line organisms within a traditional amorphous computer. These algorithms use chemical concentrations as one of the primitives for cell organization (Coore, p25). Radhika Nagpal uses gradients to organize cells in her work on folding a sheet amorphous computer into origami shapes (Nagpal, pp34-36). The gradient paradigm is an important building block of virus-programs. As a virus spreads through a network of cells, it can keep track of a virtual chemical concentration and make decisions about the cells based on that concentration. My algorithms rely on gradients as programming models.

Ron Weiss introduced the idea of messages that change the state of the cell in his work on a rule based system. The messages passed between the cells are not executable, but the cells are preprogrammed to know what state bits to change based on a set of if-then rules (Weiss, pp104-107).

While many others have shown the ability to create line organisms, which is robust at the time of creation, no one has exhibited a simple mechanism to retain the robustness of the system if cell death and malfunction are allowed after the initial creation process. Currently work is being done at MIT by Lauren Clement on repairing broken line drawings. Clement's work concentrates on a more traditional approach to amorphous computing with simple messages passed between specially programmed cells. My system demonstrates both robust creation and repair mechanisms which are based on simple programming paradigms.

3. Methods and Results

In this section I will describe in detail the virus-cell model. I will define line organisms and then describe the algorithms for creating and repairing those line organisms. I will also speak about how the algorithms performed in a simulation of an amorphous computer.

3.1 Virus-Cell Model

The idea behind the virus-cell model is to remove most of the complexity from the structure of the cell and place it in the executable virus. The model emulates a biological system of cells and viruses; the viruses insert their executable instructions into the cell's DNA and the cell replicates and executes those instructions. The copies of the virus are then forwarded onto the surrounding cells.

An amorphous computer can be thought of as a plane, or space, of randomly placed processors in a communication medium. Figure 1 shows an example of an amorphous computer. The dark cell's communication radius is shown; the cells within the radius are the only ones the dark cell knows about and can communicate with.

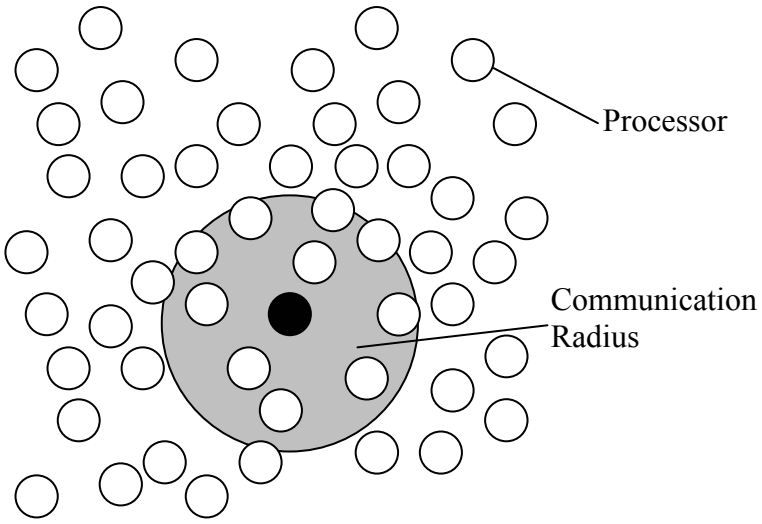

Figure 1. A sample amorphous computer is composed of randomly placed processors in a plane or in space. Each processor can communicate with its neighbors within some communication radius.

3.2 Line Organism

I have repeatedly mentioned that I designed algorithms to create and repair line organisms but I have not yet defined what one is. A line organism is a collection of lines connected in a certain way. Every line organism has a single body line. Each cell in the body line has some probability of spawning a leg. Each cell in the legs also has some probability of spawning a subleg, and so on. Therefore, a line organism is defined by the length of the body, and pairs of leg length and spawn probability numbers. For aesthetic reasons a different color may be assigned to each level of legs. An organism resembling a millipede might have a body of length 100, and a 25% change of each of those cells spawning a leg of length 5.

Figure 2 shows a sample line organism. This line organism has a body line of length 10 and four legs of length 4 each. The legs do not have any sublegs.

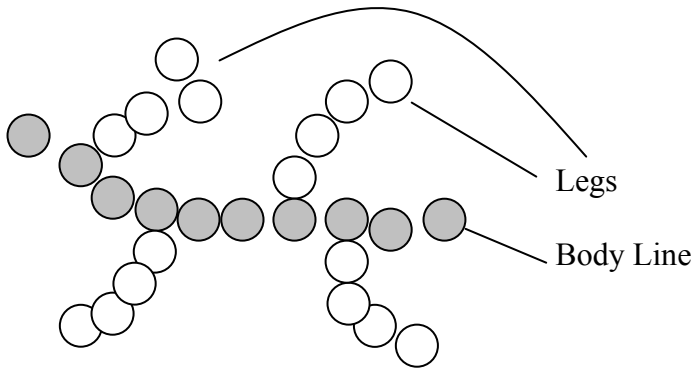

Figure 2: A sample line organism with a body line of length 10 and four legs of length 4 each. This line organism has a body and a single level of legs. Other organisms can have many more levels of legs.

3.3 Generating Line Organism Algorithm

There are 3 viruses that combine to create a line organism. The first virus is called the *line virus*, because inserting a copy of this virus creates a line organism. The basic job of the line virus is to propagate out from its source and enumerate the number of hops each cell is away from the source. The virus has an internally represented hop variable which is incremented within each cell. The virus is forward from cell to all its neighbors. The virus also stores in the cell's memory what cell sent it the virus in the first place; I call this cell the *previous* cell. Once in a new cell, the virus checks that a copy of itself has not yet visited this cell and if not, repeats the process. When the virus is created, it asks the cell to generate a random 32 bit integer. Every cell this virus visits gets a flag stored in its memory keyed on that integer. This mechanism allows the virus to determine if a copy of itself has previously visited the cell.

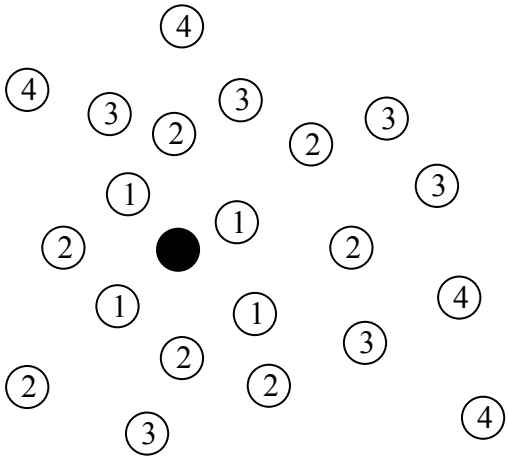

Figure 3: The state of an amorphous computer after the insertion of a line virus. The black cell is the one that started with the line virus.

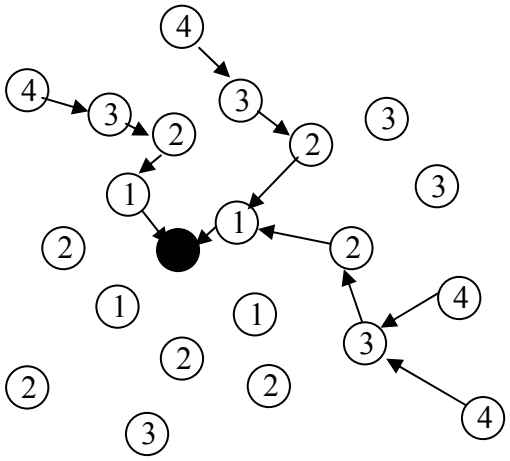

Figure 4: The state of an amorphous computer after the backtrack virus travels back to the origin. The arrows show the previous cell. Note that only the cells whose descendants reached the desired length of 4 have received a backtrack virus and are the only ones depicted with arrows.

Figure 3 shows the state of an amorphous computer after the line virus is allowed to spread. The number within each cell represents the number of hops it is away from the source. Once the desired number of hops is reached, say 4, the virus stops propagating out from that cell but mutates and now gets forwarded to the previous cell. I call this virus the backtrack virus because it traces back to the original cell and informs that cell of a possible line formation. Again, the virus records the variable which sent the virus to each cell in that cell's memory. I call this recorded cell the *next* cell. Figure 4 shows that state of the computer after the backtrack virus is allowed to propagate back to the source. Those cells that got a backtrack virus have arrows indicating their previous cells.

Finally, when the backtrack viruses arrive at the source cell, they mutate and send out a new virus, the forwardtrack virus. The virus knows that the cell is the original because it is the only one not to have a previous cell recorded in its memory. The virus sets a flag to ensure that only one forwardtrack virus will be sent. The forwardtrack virus follows the next cell links and changes the color of the cell. Changing the color is equivalent to letting the cell know it is part of a line organism, with the previous and next cells as its neighbors in the organism. The forwardtrack virus also creates new line viruses with the specified probability in each cell to create the legs. The process repeats until the entire

organism is created. As soon as the forwardtrack organism is created, the original cell starts a timer. Each time the timer triggers, a repair virus is executed. The repair virus monitors and repairs the line organism.

3.4 Repairing Line Organism Algorithm

The repairing line virus is relatively simple. It is released from the source cell every once in a while (specifically when a preset timer triggers the release). The virus walks down the organism following the next cell links. In each cell, the virus assures that cell that it is still part of the organism, resetting its color timer. A cell can determine if a neighbor cell is no longer responsive via a handshake. If the next cell is either dead or has otherwise mutated so that it is no longer capable of performing the handshake, the line organism is assumed to be broken. At this point, the repair virus knows the desired length of the line and also the number of cells that it has walked down (by keeping an internal variable), so it simply creates a new line virus with the desired length such that the overall length of the line will be the same as the original. The lost legs are recreated with the desired probability and the organism is repaired.

3.5 Algorithms in Practice

In practice the algorithms described in sections 3.3 and 3.4 worked extremely well. The nature of the virus spreading avoids obstacles like dead cells or areas of empty space. If a path of cells is chosen such that there is a dead end and not enough cells are available for the creation of the line organism, the line virus never creates the backtrack virus thus only the useful paths are explored. Whenever the environment was such that the creation and repair of an organism was possible, the algorithms were able to create such organisms.

Figure 5 shows my simulator after it has created a line organism. This organism has a red body line of length 40, blue legs of length 10 that grow off roughly 10% of the body cells, and green sublegs of length 5 that grow off roughly 5% of the leg cells.

In writing these algorithms, I found it most useful to think as if I were a cell. I wrote statements such as "when I receive the virus, I need to check if I've already been exposed and if not I should reset my timer and forward the virus to my neighbors." This mentality made the design of the viruses simple.

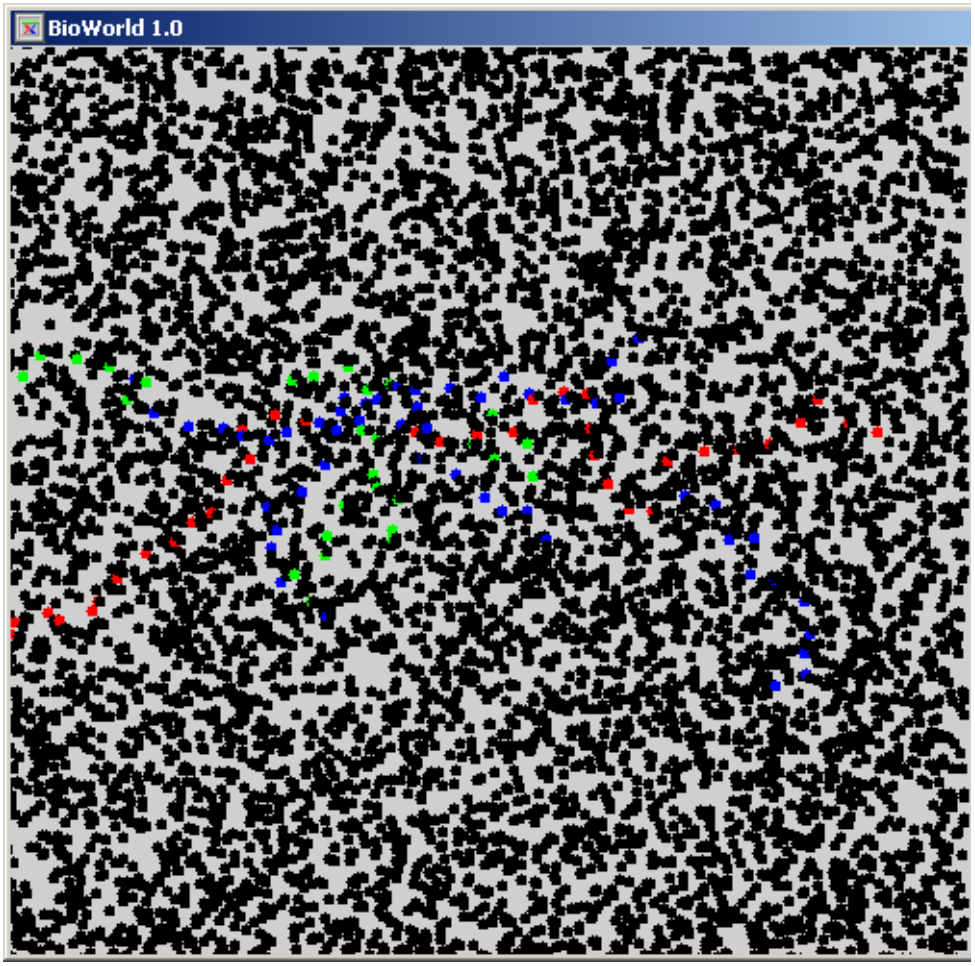

Figure 5: A sample line organism created in simulation using the viruses described in sections 3.3 and 3.4. This organism has a red body line of length 40, blue legs of length 10 that grow off roughly 10% of the body cells, and green sublegs of length 5 that grow off roughly 5% of the leg cells.

The cells described in this work were shown to be able to generate line organisms. It is fairly clear that the same cells can be used to create better defined line organisms, for example ones that specify a start and an end point and ones that have a deterministic placement of legs rather than a probabilistic one. For example, a millipede in nature has

a straight body, and symmetric legs every 2 mm or so. It is also possible to create organisms that require other shapes such as circles or polygons.

Although in simulation these algorithms create pretty pictures, the work directly relates to building at hoc networks in a system with many communicating processors, some of which may not function according to specification. The work also related directly to organizing large number of autonomous robots that have some communication power into complex and possibly recursive structures. For example, it may be useful for a group of robots on the face of Mars searching for an object to self-organize into a combing line.

4. Contributions

The goal of my work was to define a new model of thinking about an amorphous computer and to extract new programming paradigms. I defined and simulated the virus-cell model of an amorphous computer which assumes even less of hardware than the traditional amorphous computer and moves most of the algorithm's complexity into the software. Defining more reusable processors reduces the cost of manufacture. The software can be tested in simulations to iron out possible logic faults.

I demonstrated the use of the virus-cell model of computation by creating and maintaining line organisms. I have also speculated about other tasks that can be done using the same processors and the virus-cell model. Further, I demonstrated the ability to perform a new task: detecting malfunction and repairing of line organisms.

I believe that the virus-cell model is a very powerful way to think about programming a highly nonsynchronized distributed system of computers such as a swarm of robots. My future work includes giving cells actuators and allowing them to move to form complex structures. I believe that use of the virus-cell model will generate new paradigms that will promote robustness of self-organization algorithms.

Bibliography

Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight, T., Nagpal, R., Rauch, E., Sussman, G., and Weiss, R. Amorphous computing. Communications of the ACM, 43(5), May 2000.

Butera, William, *Programming a Paintable Computer*. PhD Thesis, MIT, Program in Media Arts and Science, February 2002.

Coore, Daniel, *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer*. PhD Thesis, MIT, Department of Electrical Engineering and Computer Science, February 1999.

Nagpal, Radhika, *Programmable Self-Assembly: Constructing Global Shapes using Biologically-inspired Local Interactions and Origami Mathematics*. PhD Thesis, MIT, Department of Electrical Engineering and Computer Science, June 2001.

Weiss, Ron, *Cellular Computation and Communications using Engineered Generics Regulatory Networks*. PhD Thesis, MIT, Department of Electrical Engineering and Computer Science, September 2001.