Fifth IEEE International Conference on Self-Adaptive and
Self-Organizing Systems, Ann Arbor, Michigan, October 2011.

An Evolutionary Approach to Network Self-Organization and Resilient Data Diffusion

Andres J. Ramirez, Betty H.C. Cheng, and Philip K. McKinley
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824
{*ramir105, chengb, mckinley*}*@cse.msu.edu*

***Abstract*—Data diffusion techniques enable a distributed system to replicate and propagate data across a potentially unreliable network in order to provide better data protection and availability. This paper presents a novel evolutionary computation approach to developing network construction algorithms and data diffusion strategies. The proposed approach combines a linear genetic program with a cellular automaton to evolve digital organisms (agents) capable of self-organizing into different types of networks and self-adapting to changes in their surrounding environment, such as link failures and node churn. We assess the effectiveness of the proposed approach by conducting several experiments that explore different network structures under different environmental conditions. The results suggest the combined methods are able to produce self-organizing and self-adaptive agents that construct networks and efficiently distribute data throughout the network, while balancing competing concerns, such as minimizing energy consumption and providing reliability.**

***Keywords*-Evolutionary algorithm, genetic programming, cellular automata, self-organization, data diffusion.**

I. Introduction

Data diffusion is a technique for replicating and distributing data to all nodes within a network. This process of physically isolating copies of data improves both data protection and availability [1]. Designing and implementing data diffusion techniques, however, is a challenging task that involves tradeoffs with maximizing data protection and network performance while minimizing operational costs, such as resource provisioning [2]. This paper presents an evolutionary computation approach for evolving digital organisms (agents) that self-organize into different types of network structures and distribute data to all other nodes within the network.

Techniques for data diffusion can be engineered for specific environments [2], [3], adapted from observed behaviors in nature [4], or automatically generated by evolutionary algorithms [5]–[8]. For instance, Wilkes et al. [2] proposed a data diffusion algorithm for improving data reliability and network performance in remote data mirroring environments. Similarly, Babaoglu et al. [4] developed a set of biologically-inspired design patterns applicable to frequently encountered problems in distributed systems, including data diffusion. In contrast to these approaches that either engineer a technique or adapt one from nature, Knoester et al. harnessed the process of evolution to explore algorithms for network construction [5] and data diffusion [6]. While their results demonstrated digital evolution is a viable platform for realizing self-organization and self-adaptation in distributed systems, the evolved organisms also exhibited biological behaviors, such as self-replication, that may be irrelevant within the context of a distributed system.

This paper introduces GAIA, an evolutionary computation approach that leverages an interpreted linear genetic program (LGP) [9] to evolve digital organisms that function as self-organizing and self-adaptive network controllers atop a cellular automaton (CA) [10], [11] infrastructure. The underlying CA infrastructure facilitates the exploration of self-organizing and self-adapting behaviors by restricting the interactions between organisms. In particular, digital organisms in the CA only interact with adjacent organisms (i.e, neighbors), and do not have access to global state information. Instead, digital organisms must independently perform decision-making with limited information about their local surrounding environment in order to cooperate towards a particular task.

GAIA uses a LGP to evolve digital organisms that are able to sense other physically adjacent organisms, establish communication links with them, and send and receive messages through those communication links. GAIA also uses a CA to evaluate the quality, or *fitness*, of each digital organism evolved by the LGP. This evaluation step seeds a candidate digital organism into a homogeneous CA environment, where interactions are determined not only by the state of the organisms themselves (e.g., whether they have a message to send), but also by the underlying network topology. The LGP generates new digital organisms through recombination and mutation in order to exchange and randomly modify executable instructions of existing digital organisms, respectively. This iterative process continues until the maximum number of iterations, or *generations*, is reached.

We assess the effectiveness of GAIA by conducting several experiments under different environmental conditions. The results suggest that the combined methods are able

to produce self-organizing and self-adaptive behaviors that construct networks and distribute data throughout the constructed network. Moreover, GAIA is capable of balancing competing concerns, such as minimizing energy consumption while maximizing data reliability and network performance. The remainder of this paper is organized as follows. Section II presents background material on genetic programming and cellular automata. We then introduce the proposed approach in Section III and describe how it can be applied to network self-organization and data diffusion. Next, we present experimental results in Section IV and discuss them in Section V. Section VI overviews and discusses related work. Lastly, we summarize our main findings and present future directions in Section VII.

II. BACKGROUND

This section presents background material on genetic programming and cellular automata, both of which are used as enabling technologies for our work.

A. Genetic Programming

Genetic programing is a search heuristic for automatically generating executable programs that solve specific tasks [12]. A genetic program comprises a collection, or *population*, of individuals. Each individual encodes a set of instructions and terminals that make up a candidate executable program. While genetic programming often uses a tree-based representation to encode programs, the approach presented in this paper uses an interpreted *linear* encoding [9]. This linear encoding, depicted in Figure 1, comprises a vector of instructions that can be executed sequentially from left to right. These instructions are executed by a genetic program in order to evaluate the *fitness* or quality of each individual. The fitness value assigned to an individual is proportional to how well it solves the task at hand.

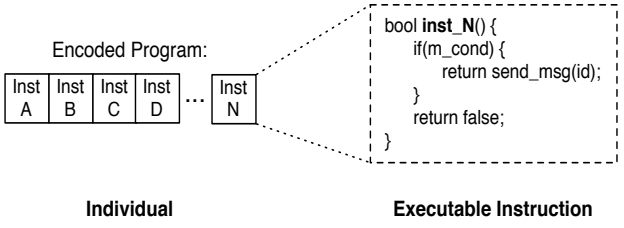

Figure 1. Interpreted linear genetic program.

A genetic program guides the search process towards promising areas of the solution space by comparing the relative fitness values of individuals. To generate new individuals, a genetic program often applies two key operators: crossover and mutation. The *crossover* operator exchanges instructions from two existing individuals to form two *new* individuals. For example, Figure 2(a) illustrates how the two-point crossover creates an Offspring I by combining the instructions $\{A,E\}$ from Parent I and instructions $\{F,G,H\}$ from Parent II. In contrast, the *mutation* operator creates a new individual by inserting, removing, swapping, or replacing random instructions from an existing individual. For example, Figure 2(b) shows a mutated genome, I′, where the position of instructions $\{B\}$ and $\{C\}$ have been swapped, and an instruction $\{R\}$ inserted (denoted by the shaded block). While crossover exchanges building blocks (i.e., important groups of instructions) between individuals, mutation adds new solution elements (i.e., diversity) to a population. A genetic program terminates once a maximum number of generations is reached or a sufficiently fit solution is discovered.

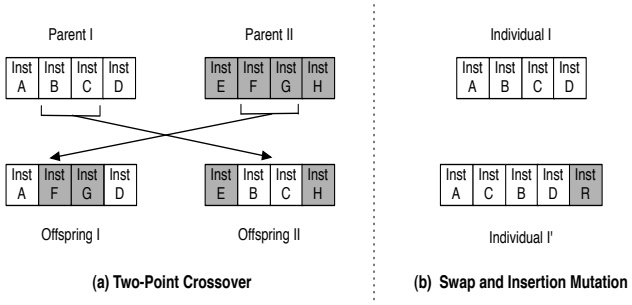

Figure 2. Linear genetic program crossover and mutation operators.

B. Cellular Automata

A cellular automaton (CA) [10], [11] is a discrete model with a finite number of cells, each of which comprises a finite-state machine. As Figure 3 illustrates, each cell in the CA is connected to a set of neighbors, or adjacent cells, as defined by an underlying topology. Furthermore, only neighboring cells interact with each other. For example, Figure 3(a) depicts a complete topology where a cell may interact with all other cells in the CA. In contrast, Figures 3(b) and 3(c) respectively depict a grid and random topology where a cell may interact with some cells but not others. Moreover, a cell's set of neighbors may be *fixed*, where it remains static as the CA executes, or *relative*, where it dynamically changes in response to the interactions between cells.

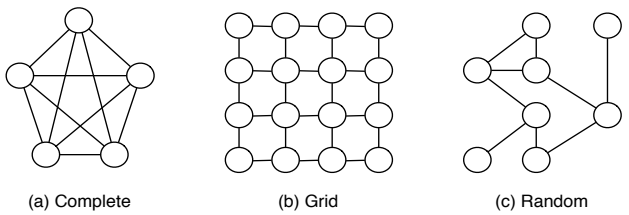

Figure 3. Examples of cellular automaton topologies.

To produce a new generation, a CA simultaneously executes the set of rules defined in each cell's finite-state machine. Which rules are applied often depend on the state of neighboring cells (e.g., a new message is available). Even CAs with a simple set of rules can produce complex interactions and behaviors. Wolfram et al. [13] classified the behaviors of CA's based on whether they produced stable homogeneous states, or complex, seemingly random behaviors.

III. Proposed Approach

This section describes how GAIA combines a linear genetic program and a cellular automaton to produce network construction and data diffusion strategies.

A. Overview of Gaia

GAIA combines an interpreted linear genetic program (LGP) and a CA to evolve digital organisms, or agents, capable of constructing and maintaining a network while distributing data to other organisms in the network. As Figure 4 illustrates, the LGP is responsible for evolving a set of candidate digital organisms, and the CA is responsible for computing the organism's fitness value. In particular, the LGP takes an individual from the population, extracts the encoded program in its genome, and then seeds each cell in the CA world with that program producing a homogeneous CA world. The CA then executes this candidate program for a specific number of generations, and computes a fitness value proportional to how well the program solved the task at hand, such as constructing a network or diffusing a message. In each generation, the LGP repeats this process for every individual in the population. The fitness values produced by the CA evaluation enable the LGP to guide the search process towards new solutions that are, ideally, better than the ones previously explored.

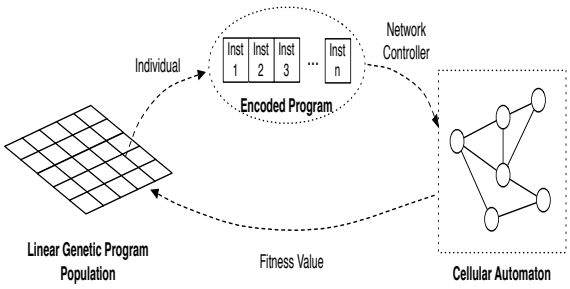

Figure 4. Combined linear genetic program and cellular automaton.

B. Description of Cellular Automaton

For this work, the structure of the CA is modeled as an undirected graph that comprises cells and edges. Each cell stores a digital organism and an identical copy of the network controller being evaluated by the LGP. The underlying graph topology, depicted by dashed edges in Figure 5, defines the set of neighbors with which a digital organism interacts. For example, organism *A* is a neighbor of organisms *B*, *C*, and *E*. In order to construct a network for sending and receiving messages, an organism must first *activate* an edge, depicted by bold edges in Figure 5, to at least one of its neighbors. As such, organism *A* can send and receive messages from organisms *B* and *C*, but not from organism *E*. Moreover, organisms are not given instinctual knowledge about their neighbors and must therefore sense their environment in order to construct networks and distribute data to other organisms.

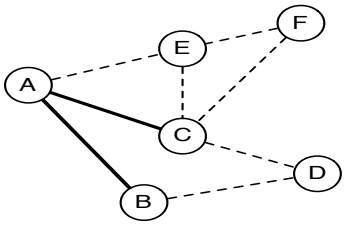

Figure 5. Underlying CA environment model.

As Figure 6 shows, in addition to the network controller being evaluated, each digital organism also comprises a fixed amount of energy, two messaging queues, a stack-based register, a set of indices to neighboring organisms, and a pointer to a target neighbor. In particular, the *outgoing* queue stores messages that the organism desires to send, and the *incoming* queue stores received messages. These messages are queued until the organism explicitly moves them. To this end, an organism may drop messages from either queue, as well as transfer a message from its incoming queue to its outgoing queue. This process of transferring a message to the outgoing queue and sending it to a neighboring organism realizes the process of data replication and diffusion, respectively, in a distributed system.

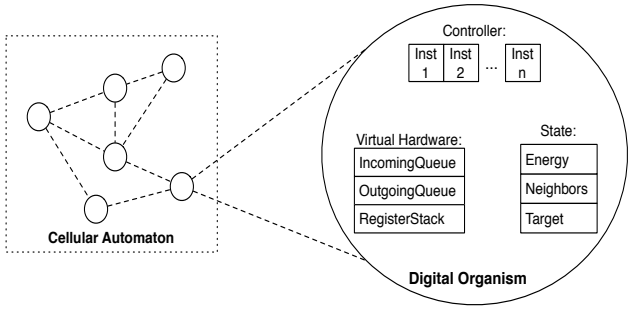

Figure 6. Resources available to a digital organism.

The CA concurrently executes the network controllers of the organisms in order to evaluate the characteristics of the resulting network construction and data diffusion strategies. Table I describes the set of instructions that an organism may use to construct a network and diffuse data. In general, these instructions enable an organism to sense its neighborhood, self-organize into a network by activating links, and send messages to other organisms in

Table I
INSTRUCTION SET FOR CONSTRUCTING NETWORKS AND DIFFUSING DATA

Instruction	Description	Cost
HasNeighbors	Returns true if organism has at least one active link to a neighbor.	2
SenseNeighbors	Returns the ID's of neighboring (connected) organisms.	5
CreateLink	Activates the link to target neighbor.	5
RemoveLink	Deactivates the link to target neighbor.	5
SetNextTarget	Points target to a neighboring organism with the next highest ID value.	2
SetRandomTarget	Points target to a random neighbor.	2
HasMessageToSend	Returns true if outgoing queue is non empty.	2
SendMessage	Places the front message in outgoing queue to the target's incoming queue.	5
IsMessageAvailable	Returns true if incoming queue is non empty.	2
HasSeenMessage	Returns true if front message in incoming queue has been previously sent by organism.	5
TransferMessage	Pops the front message in the incoming queue and pushes it into the outgoing queue.	2
DropIncoming	Pops the front message in the incoming queue.	2
DropOutgoing	Pops the font message in the outgoing queue.	2
GetCostToMessageNeighbor	Retrieves the cost to send a message to a neighbor.	5
GetNumMessagesSentByNeighbor	Retrieves the number of messages a neighbor has sent.	5
GetNumMessagesQueuedByNeighbor	Retrieves the number of messages queued at a neighbor.	5
GetRemainingEnergyAtNeighbor	Retrieves the amount of energy remaining at a neighbor.	5
IF	Advances instruction pointer by some offset if next instruction evaluates to false, otherwise flow continues to the next instruction.	2
IF NOT	Advances instruction pointer by some offset if next instruction evaluates to true, otherwise flow continues to the next instruction.	2
AND	Returns true if the next two instructions return true, otherwise returns false.	2
OR	Returns true if either of the next two instructions return true, otherwise returns false.	2
$<$	Returns true if the value of the next instruction is less than the value of the instruction that follows it, otherwise returns false.	2
$>$	Returns true if the value of the next instruction is greater than the value of the instruction that follows it, otherwise returns false.	2
NOP	Do nothing.	0

the network. In particular, the evolutionary process selects which instructions, and in what order, to incorporate within a digital organism to solve a specific network construction and data diffusion task. Furthermore, the evolutionary process may combine executable instructions with logical and conditional operators to generate more complex behaviors. For instance, combining the instructions *HasSeenMessage* and *DropIncoming* produces a behavior to avoid resending messages an organism has already distributed throughout the network. In addition, each instruction is also associated with a specific execution cost that can be configured to model different network environments.

C. Fitness Functions for Interpreted LGP

The LGP in GAIA computes the fitness value of each individual by executing the network controller within the CA and observing how well it accomplishes its tasks. To this end, we defined five fitness sub-functions to guide the LGP towards solutions that maximize data reliability and network performance while minimizing operational costs, such as activating links and messaging other organisms within the network. While the following set of fitness sub-functions require global-state knowledge about the structure of the network, these are applicable only at design-time. In particular, these fitness sub-functions enable GAIA to generate network controllers for a specific network topology that can then be deployed and executed without run-time access to global information. Moreover, this set of fitness sub-functions can also be modified to explore different aspects of network construction (e.g., efficient routing topologies) and data diffusion strategies (e.g., data routing protocols).

Network Construction. In terms of network construction, GAIA applies two different fitness sub-functions. The first fitness sub-function, $\mathrm{F_{conn}}$, rewards organisms for constructing a connected network. In particular,

$$\mathrm{F_{conn}} = \begin{cases} 100.0 & : \quad connected \\ 0.0 & : disconnected \end{cases}$$

This fitness sub-function is discrete in the sense that digital organisms either receive a full reward, if the network is connected, or nothing at all. As such, this fitness sub-function reflects the importance of constructing a connected network for diffusing data. In addition, the following fitness sub-function, $\mathrm{F_{links}}$, rewards organisms for constructing a network by activating as few links as possible (i.e., a spanning tree):

$$\mathrm{F_{links}} = 100 * (1.0 - \frac{\mathrm{Links}_{active}}{\mathrm{Links}_{total}})$$

where Links_{active} is the number of active links in the network, and Links_{total} is the total number of links that could be activated in the network. Combined, these two fitness sub-functions guide the LGP towards solutions that construct a connected network with the fewest number of links possible.

Data Diffusion and Network Performance. The LGP also applies two fitness sub-functions to evaluate how candidate solutions diffuse data in terms of maximizing data reliability and network performance. In particular, the following two fitness sub-functions reward organisms for maximizing the amount of data successfully diffused while minimizing the number of messages sent throughout the network:

$$\text{F}_{\text{mess}} = 100 * \frac{\text{Messages}_{sent}}{\text{Messages}_{max}}$$

and,

$$\text{F}_{\text{diff}} = 100 * \frac{\text{Data}_{dif}}{\text{Data}_{sched}}$$

where Messages_{sent} measures the total number of messages sent through the network, Messages_{max} is an upper bound on the number of messages an organism can send given its limited energy resources, Data_{dif} is the total number of messages diffused throughout the network, and Data_{sched} is the total number of messages that organisms had to diffuse.

Energy Efficiency. The LGP also applies a fitness sub-function to evaluate how candidate solutions maximize the amount of energy conserved while constructing a network and diffusing data. In particular,

$$F_{\text{energy}} = 100 * \frac{\text{Energy}_{end}}{\text{Energy}_{start}}$$

where Energy_{start} and Energy_{end} measure the amount of energy remaining in the cellular automaton at the beginning and end of the evaluation phase.

Overall Fitness Function. These five fitness sub-functions are combined through a linear-weighted sum, as follows:

$$\text{FF} = \alpha_{\text{conn}} * \text{F}_{\text{conn}} + \alpha_{\text{links}} * \text{F}_{\text{links}} + \alpha_{\text{mess}} * \text{F}_{\text{mess}} + \alpha_{\text{diff}} * \text{F}_{\text{diff}} + \alpha_{\text{energy}} * \text{F}_{\text{energy}} \quad (1)$$

where each α_i coefficient represents the relative importance of each concern, as measured by the fitness function itself. The sum of all α coefficients must equal to 1.0.

IV. Experimental Results

This section presents two experiments that assess the effectiveness of GAIA. Both experiments focus on how digital organisms, evolved by the LGP, self-organize within the CA in order to achieve a given task, specifically network construction and data diffusion. Table II specifies the LGP and CA configuration parameters applied for these experiments. The majority of these configuration parameters are default values used often in genetic programming as they provide good starting points if no additional details are known a priori about the execution environment and the application domain. In particular, the LGP executes for a total of 75 generations and then outputs the best network controller found so far. This parameter can be adjusted depending on how rapidly the LGP converges. If the LGP does not converge within the maximum number of generations allotted, then the maximum number of generations should be increased accordingly. If, on the other hand, the LGP converges to near-optimal values within a few generations, then the maximum number of generations can be decreased to reduce the amount of execution time.

During each generation, the LGP applies tournament selection, two-point crossover, and mutation to produce a new population. In tournament selection [14], k individuals are selected at random from the population and the one with the highest fitness value survives onto the new population. This selection process is repeated until the new population size equals 20% of the previous population size. Next, two-point crossover and mutation operators are repeatedly applied until the new population size equals the size of the previous population.

For the CA configuration, each digital organism begins with 2000 units of energy resources that are consumed as the candidate network controller executes. The amount of starting energy resources can be changed in order to simulate different network scenarios. Likewise, the CA network comprises 25 nodes or organisms, though this parameter can also be changed depending on the specific network being explored. Moreover, within this energy-constrained model, node churn and link failures occur when a digital organism depletes its energy resources. In particular, an organism that has depleted its energy resources is no longer able to send (receive) messages to (from) other organisms in the network. Furthermore, messages sent to an organism that has depleted its energy resources are dropped from the network. Lastly, for each experiment, we conduct 30 trials for statistical significance and plot the mean values, with error bars are included as applicable.

Table II
CONFIGURATION PARAMETERS FOR LINEAR GENETIC PROGRAM AND CELLULAR AUTOMATON.

Parameter	Value
Population Size	50
Max. Number of Generations	75
Crossover Rate	0.4
Mutation Rate	0.4
Selection Rate	0.2
Selection Method	Tournament (K = 2)
Number Digital Organisms	25
CA Max Number of Generations	200
Digital Organism Starting Energy	2000

A. Network Construction

The objectives of this experiment are two-fold. First, this experiment evaluates whether evolved digital organisms are able to self-organize into a specific type of network. In addition, this experiment also evaluates whether the underlying CA topology (i.e., the connections between cells) affects how digital organisms self-organize. To address the first objective, this experiment rewards organisms for constructing a connected network while also minimizing the number of active links (i.e., a spanning tree). Specifically, we set the relative weights of each fitness function as follows: $\alpha_{\text{links}} = 0.5$, $\alpha_{\text{conn}} = 0.5$, and all others equal to 0. We note that the maximum fitness value achievable by an organism that constructs a disconnected network is 50 (i.e., obtains the maximum reward for F_{links} and no reward for F_{conn}). Furthermore, to achieve the second objective, we independently evaluate this network construction task on two different CA topologies, a complete undirected graph where all organisms are neighbors, and a random undirected graph where each organism is a neighbor, on average, of three other organisms.

Figure 7 plots the mean fitness values, with error bars, achieved by evolved digital organisms at each LGP generation for this network construction task. As this figure illustrates, regardless of the underlying CA topology, the evolved organisms self-organized into a connected network while minimizing active links. The non-overlapping error bars in this figure also indicate that the LGP achieved significantly higher fitness values for the connected CA topology rather than the random CA topology. A possible explanation for the difference in both fitness values and range of error bars is that the underlying CA topology plays a significant role in facilitating or hindering the organism's ability to construct a network. In particular, an organism placed in a completely connected CA topology can exploit its high degree of connectivity with other organisms to rapidly construct a connected network. In contrast, in a randomly connected CA topology, it may be possible for a digital organism to be connected to only one other neighbor, thus limiting its ability to construct a network.

Figure 8 plots the mean number of active links throughout each LGP generation. As this figure illustrates, evolved digital organisms maintained network connectivity while minimizing the number of active links. The most common evolved strategy involved organisms first constructing a dense network, thereby obtaining 50% of their maximum fitness value, and then gradually deactivating (i.e., pruning) redundant links. Another evolved behavior combined conditional and neighbor-sensing instructions to activate a link if an organism had not already activated a link to some other organism. These strategies, and their variations, enabled organisms to construct networks with, on average, 3 to 4 redundant links. Furthermore, evolved organisms

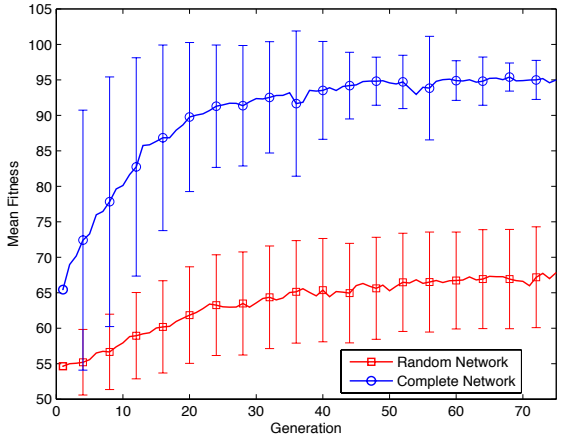

Figure 7. Mean fitness value for network construction task.

constructed spanning trees in 4 of the 50 trials for the random CA topology and in 9 of the 50 trials for a complete CA topology.

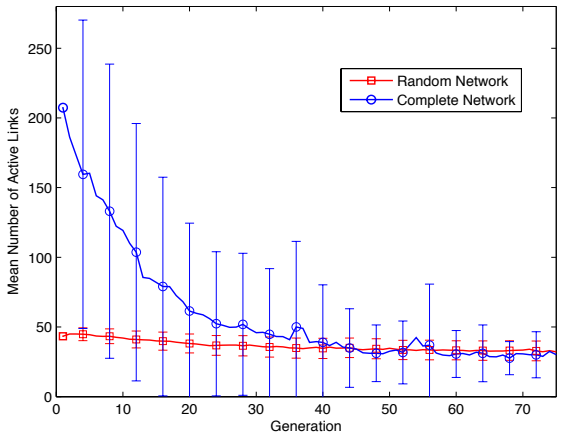

Figure 8. Mean number of active links.

B. Network Construction and Data Diffusion

The objective of this next experiment is to evaluate whether evolved digital organisms are able to construct a connected network and diffuse data. This experiment rewards organisms not only for self-organizing into a connected network while minimizing active links, but also for minimizing the number of messages sent while maximizing the number of data items diffused and the amount of energy conserved. To this end, we set the relative weights of each fitness function as follows: $\alpha_{\text{links}} = 0.2$, $\alpha_{\text{conn}} = 0.2$, $\alpha_{\text{diff}} = 0.3$, $\alpha_{\text{mess}} = 0.25$, and $\alpha_{\text{energy}} = 0.05$.

For this study, we consider data to be diffused when all digital organisms in the CA have received it. To this end, the CA inserts a unique message into the incoming queue

of 3 random individuals at different generations of the CA. These messages represent new data that must be diffused throughout the network. Figure 9 plots the mean fitness values, with error bars, achieved by evolved organisms at each generation of the LGP. As this figure illustrates, the LGP successfully evolved organisms that became more adept at constructing a network and diffusing data, as indicated by the increasingly higher fitness values.

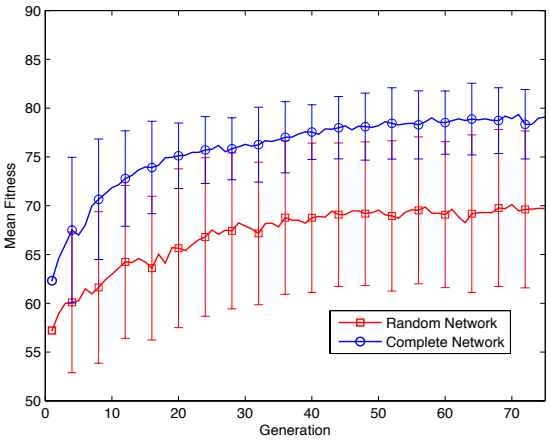

Figure 9. Mean fitness value for network construction and data diffusion tasks.

Network Construction. Figure 10 plots the mean number of active links for both the random and connected CA topologies throughout each LGP generation. This figure shows the LGP evolved digital organisms that successfully self-organized into networks that facilitated the diffusion of data. However, this figure also suggests that the LGP encountered some difficulties at minimizing the number of active links while simultaneously diffusing data across the network. In particular, for this experiment, the LGP not only had to construct and maintain a network, but also generate organisms capable of data diffusion. As a result, organisms tended to construct a dense network, diffuse data throughout that network, and *then* focus on minimizing the number of active links. The effects of this strategy are most notable for the complete CA topology, as the LGP gradually reduces the number of active links after generation 10.

Data Diffusion. In this study we consider a data item to be successfully diffused throughout the network when every organism in the CA has received a copy of the data. Figure 11 plots the mean number of data items successfully diffused by digital organisms throughout the network. In particular, organisms self-organized into a connected network and diffused data. No statistical significance can be observed in the number of data items diffused in either the complete or random CA topology, thereby implying evolved diffusion techniques were successful regardless of the underlying topology. For the completely connected CA

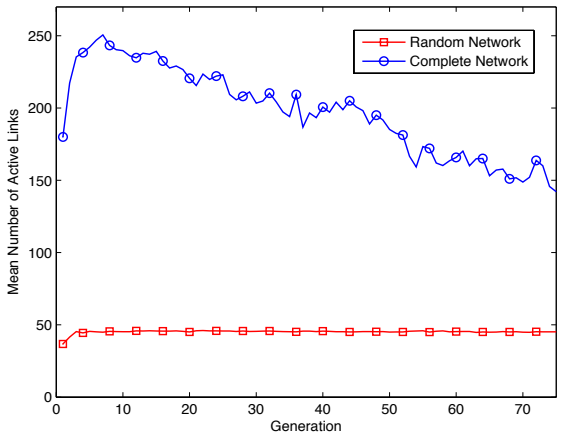

Figure 10. Mean number of active links.

topology, organisms were usually able to diffuse all three data items. In contrast, for the randomly connected CA topology, organisms were usually able to diffuse between two and three data items. As with the previous experiment, these results also highlight the relevance of the underlying CA topology on the organisms ability to not only construct a network, but also diffuse data.

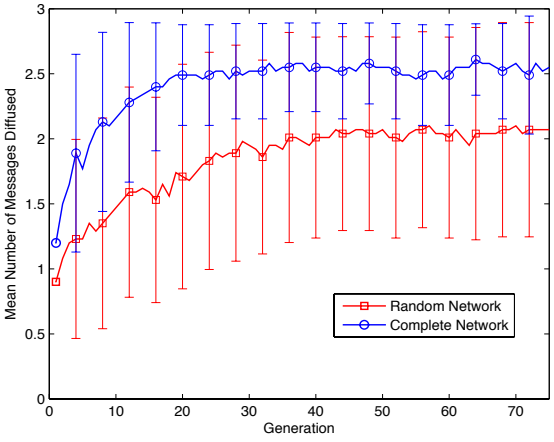

Figure 11. Mean number of messages diffused across network.

Network Performance. In addition to constructing a network and diffusing data, this experiment also rewards digital organisms for minimizing the number of messages sent through the network as part of the data diffusion process. Figure 12 plots the mean number of messages sent throughout the network at each generation of the LGP. This figure shows that digital organisms evolved by the LGP struggle to minimize the number of sent messages in order to successfully diffuse data. One possible explanation for this behavior is that each digital organism has limited knowledge

of its surrounding environment, and without global state information it is impossible for an organism to determine if other organisms have received a specific data item. As a result, evolved digital organisms adopt a strategy of sending many messages in order to maximize the number of diffused data items.

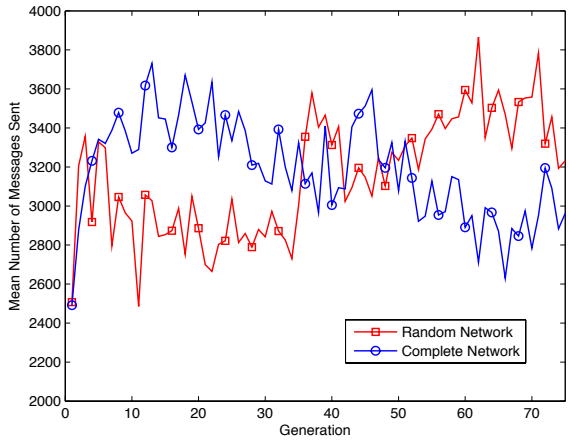

Figure 12. Mean number of messages sent throughout the network.

Energy Conservation. Lastly, this experiment also rewards digital organisms for maximizing the amount of energy they conserve as they construct a network and diffuse data. Figure 13 plots the mean remaining energy in the CA at the end of the fitness evaluation process (i.e., when 200 generations have executed in the CA). This plot shows that the LGP is able to evolve digital organisms that consume fewer resources while constructing a network and diffusing data. The LGP applied two different strategies to achieve this objective. First, the LGP reduced the length of candidate programs in order to execute fewer instructions and thus conserve energy. Second, the LGP leveraged conditionals as guards to prevent digital organisms from executing unnecessary expensive instructions, such as sending a message when the outgoing queue is empty. Combined, these two strategies enabled digital organisms to conserve more energy while reducing the number of active links and sent messages.

Sample Genome. Figure 14 presents a sample genome evolved by GAIA for constructing a network and diffusing data. As the sequence of instructions in this figure shows, this organism first executes a simple initialization phase to check for incoming messages. In particular, if no message is stored in the incoming queue, then the organism moves the instruction pointer to the end of the program in order to conserve energy. On the other hand, if the incoming queue contains a message, then the organism begins two neighbor selection phases. Both neighbor selection phases attempt the same strategy of forwarding a data item to a neighboring organism whose incoming queue is not congested with more

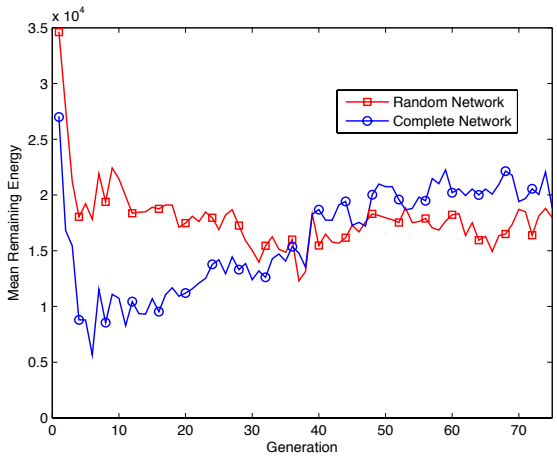

Figure 13. Mean amount of energy remaining in network.

incoming messages than it can send throughout the network. This genome comprises several logical conditionals, such as *IF-NOT*, to alter its execution path in response to the outcome of another instruction. Often, these logical conditionals are mutated into the genome around several *no-op* instructions, most likely to facilitate the conditional jumps within the genome. If no suitable organism is found in these two selection phases, then the organism selects some neighbor at random. Lastly, the organism establishes a communication link with the target neighbor, sends the message to the target neighbor, and then removes the data item from its outgoing queue.

In summary, GAIA successfully evolved digital organisms that not only constructed connected networks while minimizing active links, but also managed to diffuse data while attempting to maximize conserved energy. To achieve these objectives, most digital organisms first focused on satisfying their functional objectives of constructing a connected network and diffusing data. Digital organisms *then* addressed non-functional objectives, such as minimizing active links and maximizing network performance and energy conservation.

V. DISCUSSION

The underlying CA topology dictates, to some degree, how well digital organisms satisfy functional and non-functional requirements in self-organization tasks. Specifically, the amount of information that an organism can obtain about the network is directly related to that organism's neighborhood size. Experimental results suggest that digital organisms are able to satisfy their functional objectives despite the underlying CA topology, as digital organisms successfully constructed connected networks in both experiments and diffused data in the second experiment. The underlying CA topology, however, significantly affects how

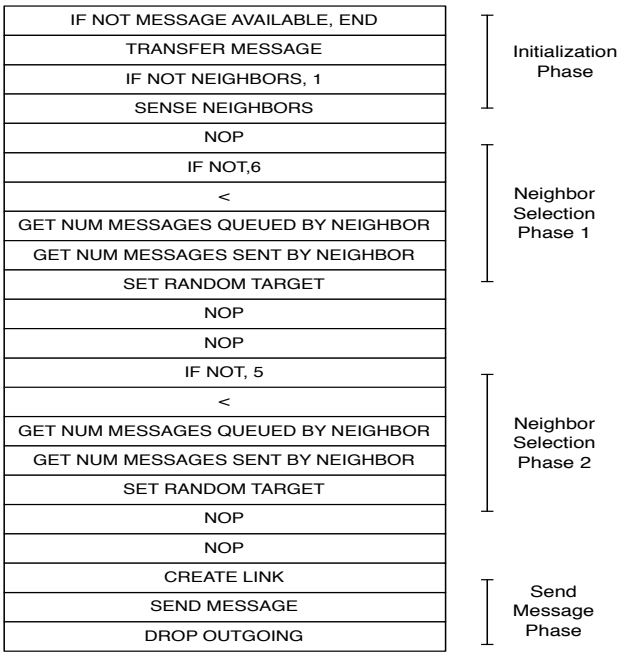

Figure 14. Sample genome for constructing a network and diffusing data.

well digital organisms satisfied non-functional objectives, such as minimizing the number of messages send and maximizing the amount of energy conserved. In particular, it is increasingly challenging for digital organisms to self-optimize their behaviors with limited information about other digital organisms in the network. These observations imply that careful consideration must be given to the underlying CA topology when modeling specific network environments.

This study applies an interpreted LGP for evolving network controllers that not only construct connected networks and diffuse data, but are also amenable to visual inspection and analysis. This simplicity in the program's representation, however, comes at the expense of modularity. In particular, true modularity was not observed in the sense of reusing a series of instructions via a subroutine or a loop. Instead, as the genome previously presented in Figure 14 illustrates, the LGP exploited the crossover operator in order to replicate important sets of instructions multiple times throughout an organism. This lack of modularity is likely caused not only by the linear representation in a LGP, but also by the lack of looping instructions in our instruction set. A more sophisticated GP platform, such as the Push GP language by Spector et al. [15] may address this issue.

VI. Related Work

This section presents related work in the areas of evolutionary computation-based approaches for network construction and data diffusion, as well as the use of cellular automata for network modeling.

A. Evolution of Network Construction and Data Diffusion

Knoester et al. explored algorithms for network construction [5] and data diffusion [6] in the Avida digital evolution platform [16]. Conceptually, Avida is a complex instance of a CA. To this end, their approach generated network controllers as the CA executed. The results demonstrated the viability of digital evolution as a means to explore self-organizing and self-adaptive behaviors. In contrast, GAIA uses a linear genetic program to *generate* network controllers that are then *evaluated* within a CA. Moreover, while the Avida instruction set includes biologically-relevant instructions for tasks such as self-replication, the instruction set in GAIA focuses only on network construction and data diffusion tasks. As a result, GAIA is able to evolve network controllers in fewer evaluations than with the Avida platform.

B. CA-based Models of Wireless Sensor Networks

CAs have been applied to explore different wireless sensor networking algorithms. These approaches tend to model the execution environment of the wireless sensor network as a fixed topology. For instance, Fan et al. [17] presented an algorithm for network self-organization that was modeled after a one-dimensional CA, such as a vector. Similarly, Cunha et al. [18] explored the viability of two-dimensional CA's for modeling sensor networks, with a focus on controlling the topology of the wireless sensor network. Fixed topologies, however, do not necessarily capture realistic scenarios that arise in networks, such as irregular topologies where nodes may fail. In contrast, Doman [19] proposed simulating and analyzing wireless sensor network algorithms with a CA whose underlying topology could change during simulation. The approach presented in this paper overlaps with the work by Doman et al. in that our CA also supports dynamic underlying topologies. However, while these approaches evaluate existing algorithms on a given CA topology, GAIA leverages the CA as a platform to automatically program self-organizing and self-adapting behaviors.

Subrata et al. [20] presented an approach that combined a genetic algorithm [14] and a CA to manage location management schemes in wireless sensor networks. Specifically, their approach evolved a vector of CA state-based transition rules that specified whether a wireless sensor node reported gathered data or not. To this end, their approach used a fixed hexagonal-shaped grid to model the placement of wireless sensor nodes. In contrast, GAIA uses an interpreted LGP to evolve a network controller that not only constructs a network from an arbitrary topology, but also diffuses data throughout the network.

VII. Conclusions

This paper presented GAIA, a new evolutionary computation approach that can be used to develop self-organizing network construction algorithms and efficient and robust

data distribution strategies. The proposed approach combines a linear genetic program with a cellular automaton to generate digital organisms capable of self-organization and self-adaptation. Experimental results demonstrate this approach is able to evolve organisms that construct different types of networks, as well as distribute messages to all other organisms in the network. Future directions for this work include exploring whether we can combine GAIA with the Push genetic programing infrastructure [15] to evolve modular network controllers.

VIII. Acknowledgements

This work has been supported in part by NSF grants CNS-0915855, CNS-0751155, CCF-0541131, IIP-0700329, CCF-0750787, CCF-0820220, DBI-0939454, CNS-0854931, Army Research Office grant W911NF-08-1-0495, Ford Motor Company, and a Quality Fund Program grant from Michigan State University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, Army, Ford, or other research sponsors.

References

[1] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes, "Designing for disasters," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2004, pp. 59–62.

[2] M. Ji, A. Veitch, and J. Wilkes, "Seneca: Remote mirroring done write," in *USENIX 2003 Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, June 2003, pp. 253–268.

[3] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE Transactions on Networking*, vol. 11, no. 1, pp. 2–16, February 2003.

[4] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montresor, and A. T. Urnes, "Design patterns from biology for distributed computing," *ACM Transactions Autonomic and Adaptive Systems*, vol. 1, no. 1, pp. 26–66, September 2006.

[5] D. B. Knoester, P. K. McKinley, and C. A. Ofria, "Co-operative network construction using digital germlines," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO. Atlanta, Georgia, USA: ACM, July 2008, pp. 217–224.

[6] D. B. Knoester, A. J. Ramirez, Betty H.C. Cheng, and P. K. McKinley, "Evolution of robust data distribution among digital organisms," in *Proceedings of the 11th annual conference on Genetic and Evolutionary Computation (GECCO '09)*, Montreal, Canada, July 2009, pp. 137–144 (Nominated for Best Paper).

[7] A. J. Ramirez, D. B. Knoester, Betty H.C. Cheng, and P. K. McKinley, "Applying genetic algorithms to decision making in autonomic computing systems," in *Proceedings of the Sixth International Conference on Autonomic Computing*, Barcelona, Spain, June 2009, pp. 97–106 (Best Paper Award).

[8] A. J. Ramirez, B. H. Cheng, P. K. McKinley, and B. E. Beckmann, "Automatically generating adaptive logic to balance non-functional tradeoffs during reconfiguration," in *Proceedings of the 7th International Conference on Autonomic Computing*, ser. ICAC. Washington, DC, USA: ACM, June 2010, pp. 225–234.

[9] M. Brameier and W. Banzhaf, *Linear Genetic Programming*, ser. Genetic and Evolutionary Computation. Springer, 2007, no. XVI.

[10] T. Toffoli and N. Margolus, *Cellular automata machines: a new environment for modeling*. Cambridge, MA, USA: MIT Press, 1987.

[11] J. V. Neumann, *Theory of Self-Reproducing Automata*, A. W. Burks, Ed. University of Illinois Press, 1966.

[12] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992.

[13] S. Wolfram, "Cellular automata as models of complexity," *Nature*, vol. 311, no. 5985, pp. 419–424, October 1984.

[14] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press, 1992.

[15] L. Spector and A. Robinson, "Genetic programing and autoconstructive evolution with the push programming language," *Genetic Programming and Evolvable Machines*, vol. 3, no. 1, pp. 7–40, 2002.

[16] C. A. Ofria and C. O. Wilke, "Avida: A software platform for research in computational evolutionary biology," *Artificial Life*, pp. 191–229, 2004.

[17] T. Fan, L. Xu, L. Yin, H. Wang, and X. Li, "A self-organization algorithm based on 1d cellular automata for networking monitoring," in *Proceedings of the 2009 First IEEE International Conference on for Networking Monitoring*, ser. ICISE '09. IEEE Computer Society, 2009, pp. 3955–3958.

[18] R. O. Cunha, A. P. Silva, A. A. Loreiro, and L. B. Ruiz, "Simulating large wireless sensor networks using cellular automata," in *Proceedings of the 38th Annual Simulation Symposium*, San Diego, California, United States, April 2005, pp. 323–330.

[19] M. Doman, T. Dahlberg, and J. Payton, "Simulation environment scenarios using cellular automata for wireless sensor network analysis," in *Proceedings of the 2009 Spring Simulation Multiconference*, ser. SpringSim '09. San Diego, CA, USA: Society for Computer Simulation International, 2009, pp. 521–527.

[20] R. Subrata and A. Y. Zomaya, "Evolving cellular automata for location management in mobile computing networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 1, pp. 13–26, January 2003.