FUNCTIONAL BLUEPRINTS:

A DYNAMICAL APPROACH TO STRUCTURE REPRESENTATION

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Nicolas Sebastian Estevez

January 2007

ABSTRACT

In engineering design, form has traditionally been specified explicitly using blueprints. This thesis explores an alternate way of specifying form built on interactions between dynamical systems. This alternate form specification is based on ideas from natural development. Inspired by termite nest building behavior, dynamic developmental systems are proposed as an alternate method to produce and represent structure designs, which when compared to the conventional blueprint method are a more robust form specification method, more adaptive, and even able to self-repair.

Developmental systems are uses here as a method of form specification and an evolutionary algorithm is the method of design chosen to explore the capabilities of these developmental systems. Evolutionary algorithms have already been widely studied and proven to be an effective method of finding solutions to tough problems, and in this work they are simply a validated tool being used.

The experiments included in this work use developmental systems with high degrees of system-environment interaction and show the importance of a subtle and often overlooked difference between two similar kinds of systems. An important distinction is being made between systems which both use feedback from the environment.

These systems are referred to as the reactive system and the interactive system. The **reactive systems** simply use environment feedback during their development, whereas the **interactive systems** not only use environmental feedback but actually form a two-way dynamic feedback cycle WITH the environment. Our control experiments are the systems with one-way feedback which have a system-environment interaction level where the system uses information from the environment during its development but does not affect the environment's dynamics. Our experiment systems

use dynamic feedback, which allows them to affect the dynamics of the environment while simultaneously the environment reacts to this stimulus, forming a two-way feedback loop which makes the system more situated in the environment. The experiments in this thesis used the evolutionary algorithms to search for systems which fulfilled the desired effect on the environment. In this case this effect is to build a structure that causes the average temperature in the environment to come as close as possible to a target temperature, which is specified at the beginning of the evolutionary run.

Both types of systems were evolved using evolutionary algorithms and those systems which used dynamic environmental feedback consistently displayed better performance.

BIOGRAPHICAL SKETCH

Nicolás S. Estévez Montero was born in Sacramento, California on March 22nd, 1981. During this time his parents, originally from Chile, were both working on their graduate school degrees at UC Davis. At the age of two his family moved back to Venezuela and by the time he was seven years old his family had moved to Puerto Rico. There he finished elementary school and attended high school, except for the 10th grade, which was done in Ithaca, NY at IHS. After that he attended the University of Puerto Rico, Mayaguez, graduating magna cum laude in 2004. Since then he has been studying at the Sibley School of Mechanical and Aerospace Engineering at Cornell University. He defended this thesis in August 2006.

To my loving family

L. Antonio, Mireya, Marcel, Claudio, and little Nicolas Hans

ACKNOWLEDGMENTS

I would like to thank the Alfred P. Sloan Foundation and the National Action Council for Minorities in Engineering (NACME), Inc. for their financial support during my time at Cornell.

Thanks to Hod Lipson for serving as my advisor, for his guidance, his wisdom, and especially for his patience in helping me find a research topic that we could work on together. My thanks also go to Nicholas Calderone for being a member of my committee, for his invaluable input on social insects and his willingness to have long idea generating discussions about the research. A great deal of my gratitude is extended to the members of the Cornell Computational Synthesis Lab group, for their great input at our group meeting discussions, input for earlier drafts of this work, technical help at the lab and also for their friendship.

I must thank Marcia Sawyer, our graduate program coordinator and mom away from home, for having all the answers and solving any administrative problems with amazing speed and efficiency, giving me a sense of security throughout my time at Cornell that I am very grateful for.

Also thanks to all my friends here in Ithaca, particularly to the graduate students from the M&AE department and all the Puerto Rican graduate students I have met here at Cornell. Special thanks to Justin Atchison who's help in the final stages of this thesis has proven invaluable to its completion.

And of course thanks go to my loving family for all their counsel and support, and for literally always being near, for some financial support, and for a lot of moral support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Termite mounds vary in size and shape according to the species of their builders, but their uniformity throughout a stable environment is striking. When, however, a species has a wide range over several different ecological zones then differences in mound architecture will be apparent when one zone is compared to another. (Harris, 35)

In nature termites build their own housing without using either blueprints or a complex thought process, yet their nest's structure shows great complexity, adaptability, and consistent functionality.

In engineering, designs are represented as blueprints which geometrically specify final structure. When such a structure is designed to perform a function, the complex design process takes into account expected environmental conditions and outputs a design that is optimized to work under these conditions. Once a design is final, structures built from a blueprint will only work optimally as long as its environment is the one for which it was designed, showing a performance drop as variation in the environmental conditions increases from the original design parameters. The blueprint approach to design requires considerable human effort and knowledge to produce a blueprint for a single design. Having structures that perform the same function under different environmental conditions would require an individual design for each environmental variation. Furthermore, a blueprint requires further effort independent of its conception to plan and execute its construction

This comparison between the biological approach to form specification and the engineering approach makes apparent that a better understanding of how termites build such complex and adaptive structures should be pursued.

Termite nest building behavior is affected by the environment, termites adapt their building patterns to fit their desired function over different environments. Using simple rules one can consistently build structures which perform a desired function even under varying environmental conditions. Considering the complexity of a termite, having thousands of them work together in unison to build these relatively large structures with such uniformity in it functions is yet another amazing wonder brought to us by evolution. The nest is the result of the activity of such a large number of individuals that individual variation is cancelled and the nest stands as the expression of population behavior (Emerson). It is thought that the complexity of a termites nest lies with the rich interaction between nest-system and environment.

Nature uses DNA as representation, evolution as the designer, and development as the builder agent. Every organism in nature is a developmental system with different levels of system-environment interaction; however, most organisms display high levels of system-environment interaction within our standards. And its is not just organisms, a termite nest can also be seen as a system composed of raw materials and thousands of termites which act as one system that develops within an environment. Developmental systems, such as termite nests, hold the answer to robust design representation. Proof that developmental systems are a successful method of representation is quite abundant in nature, which begs the question: How can we, as engineers and scientist, exploit developmental systems to build structures with functions that we desire consistently over a wide range of environmental conditions.

Dennis Haugen, www.insectimages.org   Barbara Strnadova, http://godofinsects.com

Figure 1.0.1 Termite mounds of *Amitermes Meridionalis* (Australia), these slab shaped nests are always constructed oriented North-South in their length-wise direction. Left: View along North-South direction. Right: Picture taken along the East-West direction.

1.1 Developmental Representation

In order to tap into the potential benefits that developmental systems offer an alternate way of representing designs is needed. A good way to go is with Lindenmayer Systems, or L-Systems, which were originally created to explain the growth pattern of plants (Francu). L-Systems are a formal representation of rule-based developmental systems (Lindenmayer). In order to develop, these systems need to be in an environment on which their rules are executed. These rules or axioms prompt the development of the system by effecting the placing/removal of materials or agent movement depending on current conditions. With simple rules and agents acting on local interactions, complexity comes from the system-environment interaction (Nagpal, Kondacs, and Chang). If these rules are properly designed or evolved they result in a functional structure being constructed over a wide range of environmental conditions. Given the adaptive behavior of many organisms in nature it could be

possible that with a better understanding of rule based systems we could design very robust systems, or evolve them using evolutionary algorithms as was done for this thesis. Environmental conditions are already present in the real world at the sites where functional structures are needed. The structures can be represented within the rules of a developmental system which when activated in an environment build said desired structures. Of course, unlike blueprints, the desired structure has no specified shape, but does have a desired function that it will perform by adjusting its developmental path based on environmental conditions. Aside from having superior robustness, developmental systems also offer the added advantage of including information on how to build the structures, this eliminates the considerable planning effort required in order to realize structures specified by blueprint. A strong drive for researchers in developmental systems is the possibility of using these systems for automated assembly.

1.2 Developmental Systems

In biology, development is centered on construction and self-organization. It is the production of a complex form and from a single cell. Early on the process is driven by embryogenesis (Wolpert). Morphogenesis also plays a large role of development and is the study of change in form (Bard). Both in biology and in computer science this study often focuses on system-environment interaction during development and on degrees of embodiment through perturbatory channels (Kumar and Bentley).

Once an environment is selected a system must be devised and tailored to work in that environment (Prusinkiewicz and Lindenmayer). In order for a system to work in an environment it must be able to manipulate that environment in some way and in some cases sense feedback from the environment. Feedback from the environment can

be in the form of temperature, spatial surroundings, chemical gradients, electromagnetic fields, and in the case of computer systems any properties present in the environment can be made available to the system.

Although in most cases, as part of the simulation, virtual systems are given properties which correspond to real world physical properties and virtual sensors are often given similar capabilities to their real world counterparts (Bongard and Lipson; Hornby and Pollack 1041–1048). The experiments included in this work were done in virtual environments and with software driven systems. These systems have sensor channels and are able to manipulate the environment. Because a system is at its core a set of rules and an interpreting agent, it is with these two components that the systems' behavior and environment interaction are manifested. A rule at its most fundamental level is simply a sensor state paired with an action and the agent is able to sense the environment for conditions specified by the rules and take the corresponding actions.

1.3 System-Environment Interaction

When talking about system-environment interaction one must be very keenly aware that there are varying degrees of system-environment interaction and what those different levels mean.

System-environment interaction is heavily studied even outside developmental systems. Environmental feedback and evolutionary algorithms can be combined to generate adaptive behavior in locomotive systems (Almássy and Vinkhuyzen 419-424; Bongard and Lipson 169- 176; Mahdavi and Bentley 248-255; Pfeifer and Iida 48–54; Quick et al.). Even though none of these works are about developmental systems they do support the use of evolutionary algorithms linked with environmental feedback.

However the following classification system was thought up concerning developmental systems only.
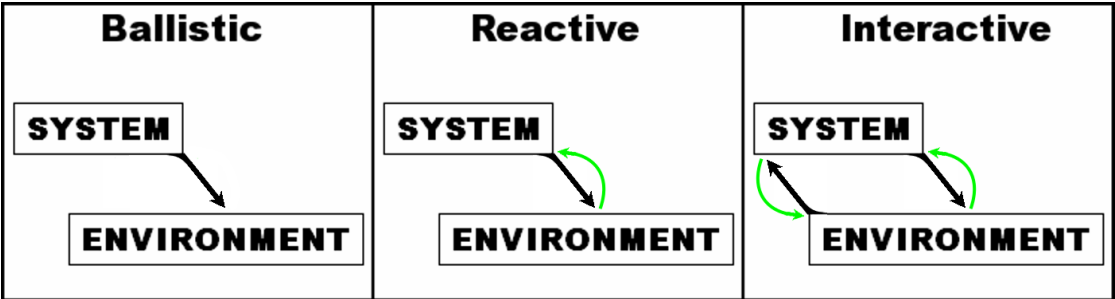
The lowest level of system-environment interaction defined here is the explicit level in which the final shape is pre-specified as is the case with blueprints. No interaction occurs between the system and the environment that would alter the final result of the systems form. This level is not included among developmental systems because in a blueprint the system does not develop, it just gets built, independently of the environment. For a system, being situated in the environment by itself entails that the systems' actions will affect the environment.

At the **ballistic level** the system develops without any kind of feedback from the environment and no knowledge of it, affecting the environment but not being affected by it. An example of this level of system-environment interaction would be a system operating with an open-loop controller that takes no input and does not make any adjustments to it behavior. As developmental systems they already offer advantages over conventional blueprints as a method of form specification (Rieffel and Pollack).

At a **reactive level** we have systems that have knowledge of their environment and use that information in their development using feedback during their development but not really affecting environment dynamics. An example of this level of interaction would be a closed-loop controller. Even though this controller uses feedback during operation it is mostly just aware of the direct effects of its own actions and is not aware of changes in environment dynamics and much less able to reach to them, such a case would occur when the system develops on a timescale that is much faster than the environment dynamics so the system is not aware of these dynamics, or in cases were the environment is not dynamic (Bentley and Clack 118-127; Eggenberger 205-213; Hemberg and O'Reilly). At this level of system-environment interaction the system is already able to adjust its developmental path to suit different environments.

However during its developmental stage the system would not be able to exploit the dynamics of its own interaction with the environment.

At the **interactive level** a system uses dynamic environmental feedback it then becomes very highly situated in the environment. This system is then able to exploit the dynamics of system-environment interaction (Nagpal). During its developmental stage, like in all levels of system-environment interaction, the system perturbs the environment through its actions. Since at this level the system is highly situated the environmental dynamics can affect the behavior and development of the system. This mutual perturbation allows the agent to be aware of the effect that its own development has in the environment and therefore use environmental dynamics to its favor. In essence the system is using dynamic environmental feedback by constantly updating its knowledge of the environment and reevaluating its actions in each execution. The interactive level is only possible to achieve when the timescale of the environment dynamics is faster or comparable to the time scale of the system's development. Systems with identical rules of behavior could potentially fall under the reactive or interactive levels of system-environment interaction based solely on a difference in timescales.

Figure 1.3.1 Levels of System-Environment interaction showing perturbation as a straight arrow and feedback as green curved arrows.

1.4 Hypothesis

The hypothesis of this thesis states that when developmental systems with an interactive level of system-environment interaction have the same performance as systems with a reactive level of system-environment interaction when compared for robustness as a method of representing structures. This robustness comes from being better able to adapt their building pattern for different environmental conditions such that their performance is averaged over multiple environmental tests.

The experiments in this thesis are evolutionary runs of interactive systems and reactive systems. In both cases the system are given the same task to fulfill and are tested under the same environmental conditions.

Another set of experiments was done to test the self-repair capabilities of the evolved systems. This set of results is discussed in chapter 7.

1.5 Experimental Approach

The concepts being explored in this work are relevant for any structure that needs to perform any function. However these are very broad concepts and the following clarification should be taken into consideration while reading the rest of this work. The uses of the words "structure" and "function" in the context of the experiments and discussions in this thesis will at times refer specifically to static structures whose function is to maintain a specific temperature.

The experiments included in this work where designed to find out if constructional processes that use dynamic environmental feedback are more robust than those that

use non-dynamic environmental feedback. Under the given level system this is a comparison of Reactive vs. Interactive systems. Previous work has been done comparing Ballistic vs. Reactive systems (Hornby 569-587) although this terminology was not used. These experiments are all evolutionary runs using a hill climber evolutionary algorithm.

Our results show that structures built using developmental systems can perform their intended functions over a range of environmental conditions. This is because a developmental system does not explicitly define the final structure, but through the execution of its rules using dynamic environmental feedback will adapt its building patterns so that the final structure will perform its functions. Therefore under different environmental conditions the final shape will vary so that the function does not.

In this thesis we set out to both point out the sometimes disregarded difference between interactive systems, which use dynamic two-way environmental feedback, versus reactive systems, which use one-way environmental feedback, and the advantages that interactive systems have over reactive systems.

1.6 Structure of this Thesis

This thesis is broken down into 8 chapters. These next three Chapters, following the introduction, explain the work that was done as well as the experimental procedure. They are broken down as follows:

- Environment and Physics: Experimental setup concerning simulation and simulated physics dynamics
- The Systems: Rules and Representation: Description of the reactive and interactive developmental systems conceived for this work.

- Evolution: Hill climber used and the structure of a fitness evaluation.

Chapter 5 goes over the main experimental results, which concerns the hypothesis of this thesis. Further results are presented in chapters 6 and 7. While the conclusion and some closing thoughts are within the final chapter of this thesis.

CHAPTER 2

Environment and Physics

When exploring subtle differences between levels of system-environment interaction it becomes critical to have well defined systems and environments. This chapter will describe the environment that was used in all the experiments discussed in this thesis along with an explanation of the systems that were tested, including how rules are represented and how the agent executes them.

A developmental system cannot materialize itself without an environment to embody it in much the same way as software cannot manifest itself without hardware to run it (Kumar and Bentley). A system can be defined as a set of rules without an environment, but an environment is required in order for these rules to develop. Just to make sense of what the rules mean there needs to be a medium in which these rules are designed to work in. Therefore it is logical to define the environment first because regardless of whether or not it was the chicken or the egg it is clear that the dirt, oxygen, and sun came before them all. In order to understand the system one must first understand the environment in which the system is to be situated.

2.1   *Environment Groundwork*

The environment in which our system will be situated in is a two-dimensional square grid composed of square cells. Each square cell has a floating-point temperature value. The temperature of a cell determines the color of that cell, going through the following color sequence, black-blue-cyan-yellow-red, starting at zero for black and one-hundred degree units being red. Temperatures are allowed to go outside this range but

they are displayed as the color for the nearest defined value. Red for values above one-hundred and black for values below zero. However temperatures stay within the range of zero to a hundred in most relevant cases, therefore the color gradient is kept constant to ease comparison between results. These are an arbitrary values being called temperature they do not correspond to any real-world units like degrees Celsius or Fahrenheit but, because is it being pictured and treated as temperature, they will be referred to as degrees or degree units.
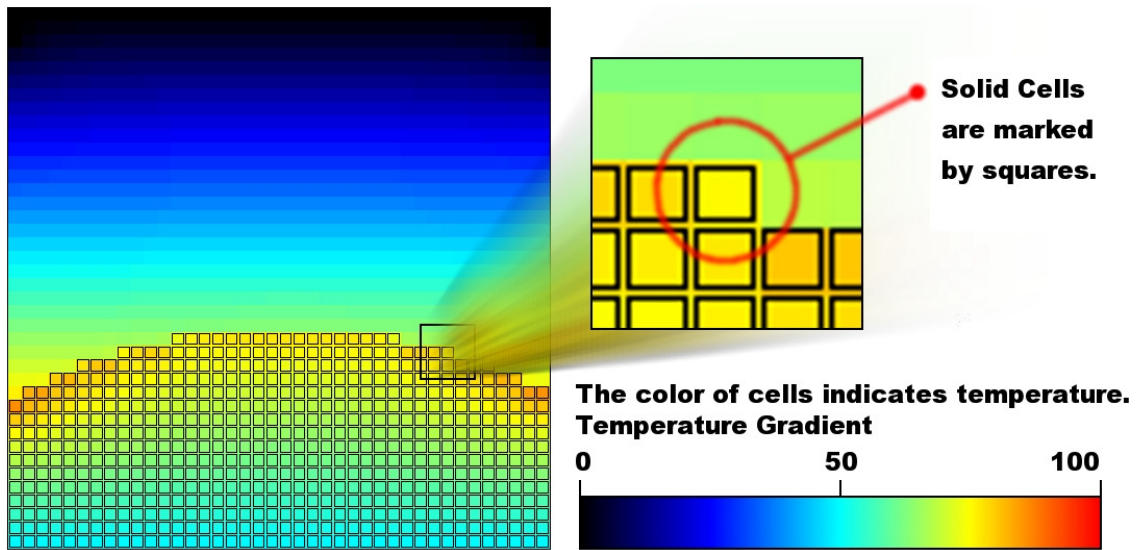
Figure 2.1.1 Screenshot of environment showing the initial conditions of matter that is referred to as the "hill" and one of the initials conditions for temperature.

2.2   Constraints and the Scenario

Cells have two possible matter states. The two matter states are solid cells and gas cells (or empty cells). The arrangement of cell states around the environment and their interaction with the physics determine the temperature distribution. The fitness of a system is calculated based on cell temperatures.

These core components are used together to build a rough representation of the environment in which a termite would built its mound, as per the inspiration of this project. Our simplified version of this environment includes a static sun and ground with air above it. In this environment our system is able to change matter states in the cells in order to build a structure, dig a hole, or in some cases both. Due to the sunray-matter interaction, these redistributions or matter caused by the system will in turn affect the temperature gradient throughout the environment. And in turn the temperature values among all the cells at the end of each test will determine the fitness value for that agent.

Our experimental area can be seen as a two-dimensional slice of a real world environment showing a side view of this environment. The bottom area on the environment grid is occupied by solid cells which represent the ground and have the shape of a rounded hill on its top boundary. Its sides and bottom boundaries are at the boundaries of our environment. The point being to picture that the ground would continue beyond these environment boundaries and we are just looking and a small piece of a larger world.

In the upper area of the grid above this "hill" there are gas cells or empty cells which represent air or the atmosphere. In keeping with the picture of this being a side view the sunrays always originate from the top of the environment sunrays always run parallel to each other but their overall angle is determined by a input value for that function. In order to avoid having the sunray code indefinitely raise the temperature of the whole environment two rows were made into "heat sinks". The top row, which represents the sky, is reset to zero degree units after every execution of the sunray code. The bottom row is also reset but always at fifty degrees. Both the sunray code and the row temperature reset code are contained within the function that applies the constraints on the environment.

2.3   Matter and Physics

The differences between the two possible matter states in cells drive the dynamics of the environment because the physics interactions totally depend on matter state. Solid cells stop sunrays and absorb heat from them while empty cells do not directly interact with sunrays and the heat transfer coefficient between cells depends on what the states of the cells are.

A physics time-step updates the temperature in each cell by applying a conduction equation with its four neighboring cells. Therefore, the temperature of each cell is updated using the difference in temperature with its neighboring cells and a conduction coefficient. There are three possible conduction coefficients. One for each of the following cell boundary types: Solid-Solid, Gas-Gas, and Solid-Gas (same as Gas-Solid). These values are specified in a symmetric look-up symmetric matrix used by the conduction code [appendix B].

Sunrays in this environment behave in a simplified way from their real-life namesakes. They travel in a straight line and collide with solid objects. Upon any collision a sunray is fully absorbed and the cell which stopped it experiences a temperature increase whose value is a pre-specified constant for added simplicity.
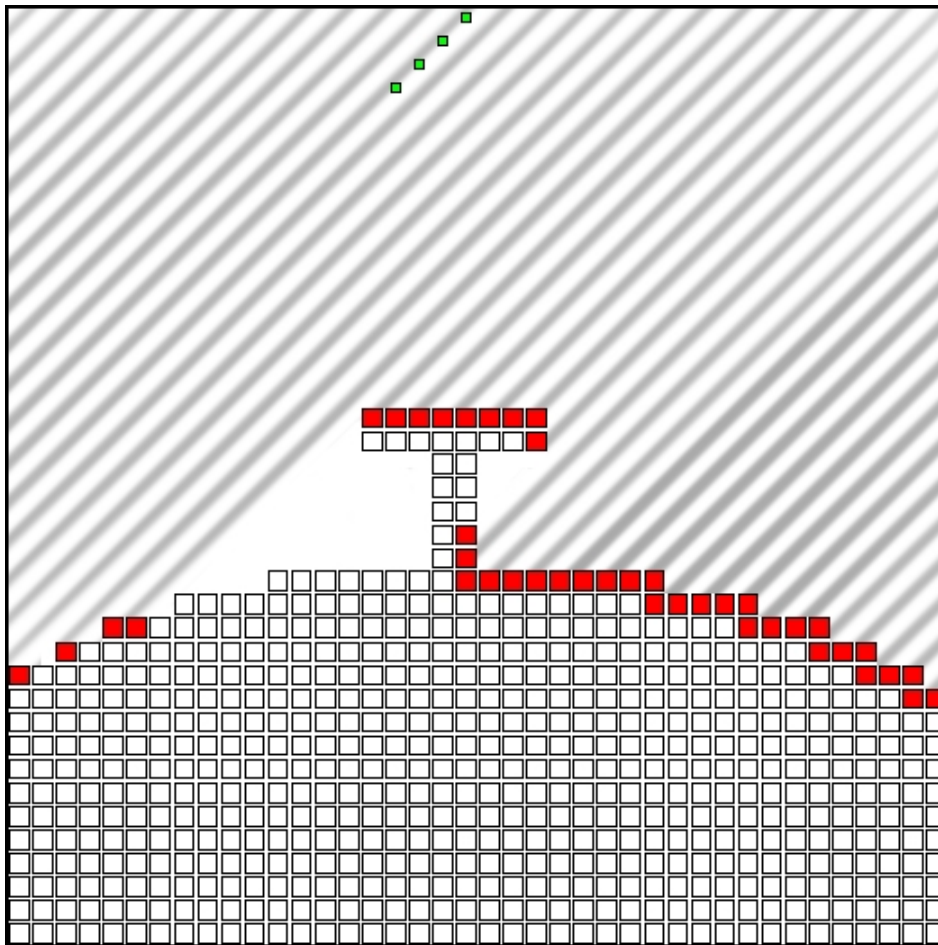
Figure 2.3.1 Sunrays with slope of 1 descending on example arbitrary structure. Solid cells that stop rays are shown in red. Small green squares at the top indicate the direction of sunrays.

CHAPTER 3

The Systems: Rules and Representation

Now that the environment has been defined, the system can also be described and understood. The word system, as it applies to this work, refers to an entity that develops by following a set of rules and has mechanism that applies those rules to the environment. Within an experiment the system would be the structure of solids blocks which develops following a set of rules. A particular instance of the system is called an individual. An individual is a specific rule set and can be evaluated to be assigned a fitness value. The rules are like the genotype of an individual and the resulting developing structure is the corresponding phenotype.

3.1 Agent and Rules

An agent is though of as being the executioner of rules. As a comparison to real life, the system could be a building that is constructed by a programmable robot, while an individual would the resulting building from a specific set of construction rules. Now if one would two models of this robotic system and given each its own set of rules then one would have two different genotypes, which when put to work could each be given it own fitness value. But if you were to program both robots identically then you would not build two individuals but two instances of the same individual, which then has only one fitness value.

During system development the agent works in steps or iterations. In each step the agent scans the whole environment looking for allowed action sites. Action sites are cells in which rules can be evaluated an executed. Action sites cannot exist within

solid cells, only empty cells. However the agent can only evaluate the rules in those empty cells which are adjacent to solid cells, which would be the surface of the solid structure. This behavior was designed to mimic how termites would build their mounds by crawling around the surface and laying or removing material. Although in the experiments the agent can scan the entire surface and then decide to take one action at a time, always choosing based on the rules.

*3.2 Interpreting the Rules*

A rule is made up of five genes. The genes are numerical values that define the behavior of the interpreting agent and thus the behavior individual. They are referred to as genes because the rules are the genotypes of our individuals. Like their biological namesakes these genes contain the information that defines an individual's development.

The first two genes are sensor genes and the last three genes specify an action [see figure 3.2.1]. As the agent scans the surface of the structure, evaluating each action site, it senses two parameters that it uses to decide in what cell it will execute an action. The first is the mean temperature of the eight cells that surround it. Next it senses the matter density of those same eight cells by taking the number of solid cells and dividing it by eight. This fraction is normalized to one-hundred, therefore having eight solid cells around it means a density of 100 and zero density would result if the agent where floating in mid-air, which never happens because sites in mid-air are not evaluated. The third value is a vector that points in the hottest direction of the local temperature gradient in the same eight cells.
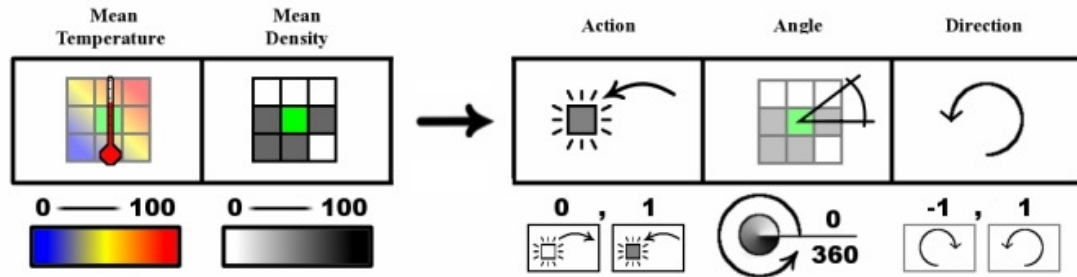
Figure 3.2.1 Rules have mixed representation. The first two genes are floating point values between 0 and 100. Out of the other three genes one is binary, another one is floating point from 0 to 360, and the last gene is also binary but with the values negative one and one.

The following is a graphical description that's equivalent to the method that the agent uses to select a cell as an action site. First the agent plots the two sensor genes of each rule in a two dimensional space, having one point per rule. Second, the agent takes the value of mean temperature and the value for matter density at each potential site and plots them, in turn, in the same two-dimensional space with the sensor gene points. As the agent checks allowable cells or action sites it ads the distances from the cell's sensor point to all the rule points as a total distance. It then selects the cell whose mean temperature and matter density sensor data has the smallest total distance to the rules, because this is determined as the best match for the rule set.
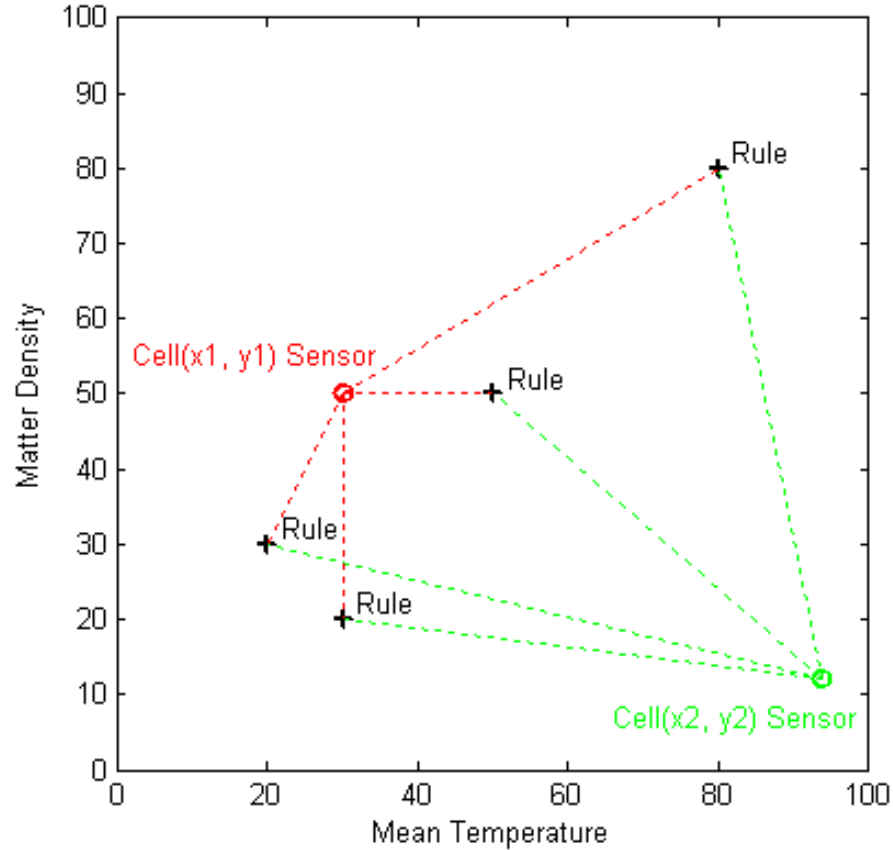
Figure 3.2.2: Graphic example of action site selection for individual with four rules. Figure shows distance measures for two cells at different environment locations. The cell at coordinates (x1, y1) is a better match that the cell at coordinates (x2, y2) because its total distances to the rule points is shorter.

The action taken by an agent is determined by the last three genes in each of the rules. Once and action site is selected by the agent the action to be taken is determined by combining all the rules by a sum of weights. As seen in figure 3.2.1 the last three genes of a rule define the action to be take and how. The first action gene is a binary gene that can have a value of 0 and 1. If the value is 0 the rule favors the remove action. And if the value is 1 the rule favors the deposit action. The word favor is used because a rule never acts alone. This is because of the sum of weights method being used. When an action site is selected by agent using the distance method describe in

figure 3.2.2, the inverse of the distances to each rule are saved as normalized weights. Therefore, the rules whose points are the closest to the sensor data point will have more weight in the actual action taken. When deciding whether or not the action will be remove or deposit the agent round the weighted average such that if this value is above or equal to 0.5 the action taken is deposit (turns an empty cell into a solid cell) and for values below 0.5 the action taken is remove (turns an solid cell into a empty cell).

The last two genes determine how the action taken is performed. The angle gene indicates, as it name suggests, at which angle the action is to be taken. This angle determines which of the eight surrounding blocks is going to be tried first. If the action taken is deposit and the block tried first is already solid then the agent tried the next block over in either the clockwise or counter-clockwise direction. This is determined by the last gene, called direction, which is also binary genes but with be values 1 and -1. For the angle genes the sum of weight adds up the vector from the rules. The vector for a rule is at the angle of that rule's angle gene and the length is determined by the weight. For the direction genes the agent simply checks the sign of the weight average and uses that value.

3.3 Fitness Function

The fitness function used for this research checks the absolute temperature difference that each cell has with the given target temperature and then averages this value. Having this value be zero would be a perfect score with the fitness becoming worse as this average distance becomes larger. In order to have an ascending fitness function the average difference calculated is then multiplied by two, for no other reason than to increase resolution and scale fitness to one-hundred, and subtracted from one-hundred.

20

So a perfect fitness would be 100, meaning that all cells are at exactly the target temperature, and then is goes down from there.

3.4 Two Test Types

When dealing with a hypothesis that considers the effect of a particular difference between two systems one needs to test this using experimental data and one will certainly be required to conduct two types of experiments. One set is presented as the control data (reactive) and the other as the experimental data (interactive). Here follows the descriptions of these two kinds of experiments.

The control runs correspond to the system that uses feedback but does not affect the dynamics of the environment. For these experiments the environment is set at the initials conditions. And the system is run for a given number of steps WITHOUT running the physics after each step. This means that while the system develops there are no updates on the temperature values in the environment. Once the system is finished the physics are run until stability and the fitness of the structure is evaluated.

The experiment runs are the ones in which the system is using two-way feedback affecting the dynamics of the environment while the environment behaves like a separate system which is also reacting to feedback given by the systems actions. For these experiments the environment is set at the same initials conditions as for the control experiment. The system is run for the same given number of steps but now this is done while running the physics after each step. This means that while the system develops the physics are being update and the temperature values in the environment will change due to the changes being made by the system. Once these cycles are done the physics are already at stability and the fitness of the structure is evaluated.

In both these test types the fitness will be solely based on the final structure. Although it is very import in both cases how the system arrives at the final structure, given the same final structure is reacted under the different tests, it will yield the same fitness. It is important to remember that the claim of the work does not ask whether one type of system can find a better path to a given goal, but whether one type of system can build better structures.

3.5 Interactive Dynamics Index

Because the relation of timescales between the system and the environment is what determines whether a system is reactive or interactive a metric needs to be defined in order to categorize these systems. Indeed there is a continuous scale from a fully reactive system to a fully interactive system.

A fully reactive system (control systems in this work) is a system that develops in a relatively static environment, meaning that the system development is instantaneous and from the systems point of the system the environment IS static. On the opposite end of the spectrum are fully interactive systems (experimental systems in this work) a system that is constantly developing in an environment that is in steady-state, meaning system development is so slow that after each system step the environment reaches steady-state before the system takes the sensor input for the next step.

An interactive dynamics index (I.D.I.) is proposed that goes from zero to one. Fully reactive systems are considered completely non-interactive so they correspond to zero dynamic interactivity, while fully interactive systems would have an index value of one. In computer simulations it is possible to create and classify fully reactive and fully interactive systems.

In practical real-world systems it would be impossible to devise a system with exactly zero or one dynamic interactivity due to the required speed of development, infinitely fast or infinitely slow. However, one must be able to determine if a given system is reactive or interactive. Where is the midpoint or system response time boundary where a system changes from being reactive to interactive?

By specifying an environment steady-state tolerance one is able to experimentally find a system response time where the system barely allows the environment to reach steady-state according to that tolerance, call this T_{ss}. A system is then considered to be interactive if its response time is one-quarter or greater this value. If this value is called T_b and then a plot of $-\log_2(1-t)*T_b$ is made, one will note that the system response time value of T_b on the y-axis will correspond to an I.D.I. value of 0.5 on the x-axis. This plot maps response times on the y-axis from zero to infinity to the Interactive Dynamic Index on the x-axis from 0 to 1, Figure 3.5.1. The previously found T_{ss} value will also always to correspond to and index of 0.9375.
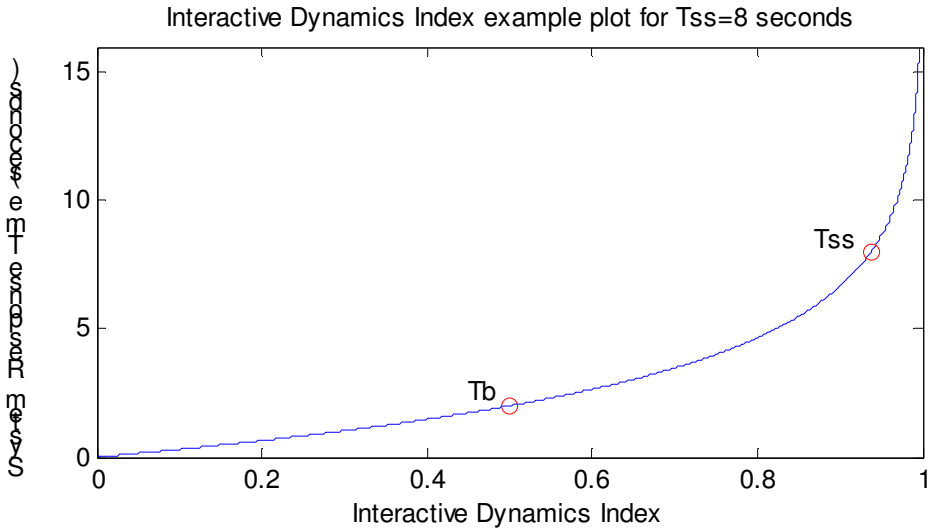

Figure 3.5.1 Interactive Dynamics Index example plot.

CHAPTER 4

EVOLUTION

The system optimization method used to test the developmental systems described in the previous chapter are Evolutionary algorithms (or EA's). These algorithms use concepts from evolution like natural selection, genetic representation, mutation, fitness in order to find the best solution to a problem.

Evolutionary algorithms have been widely used in research and design due to their ability to find solutions to hard problems (Koza). For this work it is especially fitting to use EA's as an experimental tool device since the developmental systems being used are nature inspired structure representations and an evolutionary algorithm is a nature inspired design method.

The evolutionary algorithm used in this thesis was a hill climber. A hill climber algorithm evolves a population by trying out random mutations and only keeping them if they are beneficial.

4.1   Hill Climber Algorithms

Hill climber evolutionary algorithms are classically used with a population of only one individual. The alternative of running a larger population is called the parallel hill climber and is equivalent as running several single populations runs since the individuals do not interact with each other.

The hill climber algorithm used here is of the single population variety. The initial individual was randomly generated. Random mutation was used to generate a new individual that was then evaluated and if this individual scored a higher fitness than its

single parent then it would replace the parent and the cycle would continue. This cycle would repeat itself for each fitness evaluation.

4.2   Evaluation Hierarchies

The following is a listing of the components of a fitness evaluation organized by hierarchy. Components of components are listed in parentheses next to the names.

Fitness Evaluation (Tests)

A fitness evaluation is composed of three different tests in which the variant is the angle of the incoming sunrays. Each test runs for a given number of simulation steps, at the end of which the fitness of that test is evaluated. The fitness assigned to the agent is the average fitness of the three test runs.

Test (agent actions, environment updates):

For each test, the environment is initialized with the same matter pattern every time and a given temperature distribution that corresponds to the steady-state conditions for each of the 3 possible sun angles. The number of simulation steps in a test determines how many agent actions a system is allowed before it is stopped and evaluated. Environment updates are also performed within a test but their use varies between the control runs and the experimental runs.

Agent Action:

An agent action or step consists of scanning all valid action sites. These actions sites are the cells where the agents can exist (empty cells with adjacent solid cells). In each of those evaluated cell the agent senses the average temperature and matter of the

eight cells that surround that cell and using this data and its rules the agent chooses the location where it will perform an action and execute the rules.

Environment Update (conduction physics, apply constraints):

And environment step updates the temperature values that change as a result of the agent's actions. Each environment step is broken down into a conduction physics update and the enforcement of the constraints.
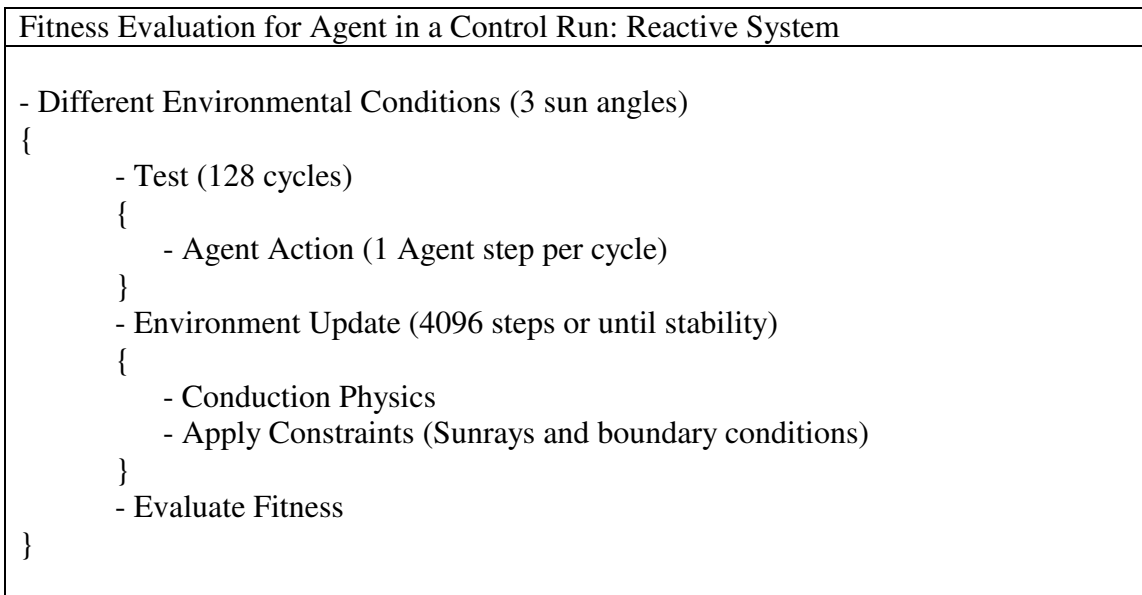
Conduction Physics:

A physics time-step updates temperature in each cell by applying the basic conduction equation with its four neighboring cells. The temperature of each cell is updated using the difference in temperature with its neighboring cells and a conduction coefficient. The conduction coefficient is dependent upon the matter state of the cell and its neighbors. These coefficients are stored in a symmetric two-by-two matrix. As mentioned before there are three heat transfer coefficients. One for each of the following cell boundary types: Solid-Solid, Gas-Gas, and Solid-Gas (same as Gas-Solid).
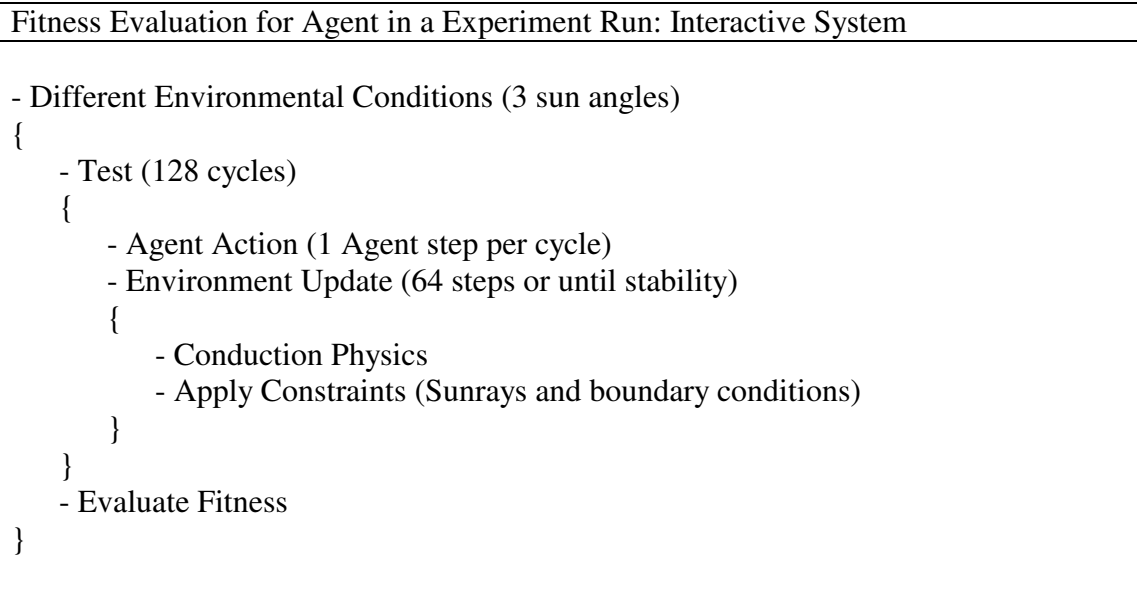
Apply Constraints:

The apply constraints function contains the sunray code which scans the environment area with rays that emerge from the top of the grid angled at a specified slope. Sunrays stop when they hit a solid cell and the temperature of the cells which stop sunrays is increased by a fixed amount in an approximation to how the sun would heat up terrain. This function also sets the temperature of the top row of cells to zero degree units and the bottom row to 50 degree units. These rows act as heatsinks to the sunrays.

4.3   Control and Experiment Systems

There are two kinds of runs that where done with the hill climber algorithm, the control experiments with one-way feedback and the two-way feedback runs. The fitness evalution for each type of run has one of the following structures of several nested for-loops.

```
Fitness Evaluation for Agent in a Control Run: Reactive System

- Different Environmental Conditions (3 sun angles)
{
        - Test (128 cycles)
        {
            - Agent Action (1 Agent step per cycle)
        }
        - Environment Update (4096 steps or until stability)
        {
            - Conduction Physics
            - Apply Constraints (Sunrays and boundary conditions)
        }
        - Evaluate Fitness
}
```

Figure 4.3.1 In control runs the agent would perform all its agent action steps without any environment updates. Following that, the environment update would run until stability followed by a call to the fitness evaluator.

```
Fitness Evaluation for Agent in a Experiment Run: Interactive System

- Different Environmental Conditions (3 sun angles)
{
    - Test (128 cycles)
    {
        - Agent Action (1 Agent step per cycle)
        - Environment Update (64 steps or until stability)
        {
            - Conduction Physics
            - Apply Constraints (Sunrays and boundary conditions)
        }
    }
    - Evaluate Fitness
}
```

Figure 4.3.2 In experiment runs the only variation is that environment updates are run after each agent action step so the agent is aware of the effects of it actions and is able to use dynamic environmental feedback.

4.4   *Walkthrough of Fitness Evaluation*

As seen in section 4.3, a full fitness evaluation for both types of runs consists of 3 tests, which in turn consist of 128 Agent Actions. The 3 tests are evaluated separately but the agent's fitness is the average of the 3 tests. The difference between the two is when the environment updates occur.

    The following is a walkthrough of one full fitness evaluation. There are three sets of animation frames corresponding to each of the three tests. The variable in the tests is only the angle of the sunrays and the same 3 angles or tests are done throughout all the experiments in this work and are the following.

| Test 1 | Test 2 | Test 3 |
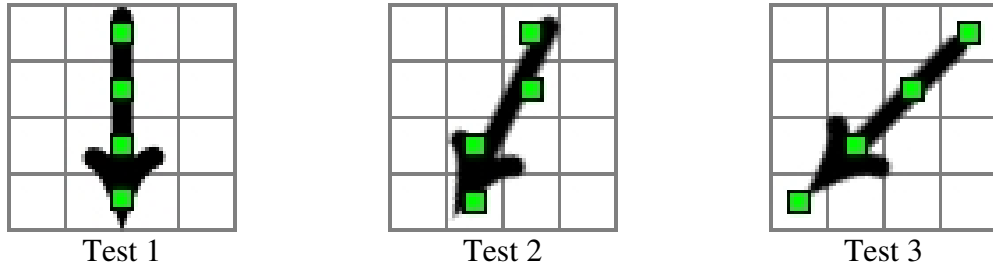
Figure 4.4.1 Arrows indicating the direction of the sunrays in each on the three tests. The green squares overlaid on top of the arrows match the arrangements drawn at the top on the environment display during each test.

The genome of the individual whose development is shown in the following

walkthrough is shown on Figure 4.4.2

Mean Temp.	Mean Density		Action	Angle	Direction
12.7226	41.7744	→	1	187.653	-1

Figure 4.4.2 Genome of Interactive System with Fitness of 68.8917, best performance found among Interactive systems with 1 rule in all 20 runs.

0 Steps	16 Steps	32 Steps
48 Steps	64 Steps	80 Steps
96 Steps	112 Steps	128 Steps
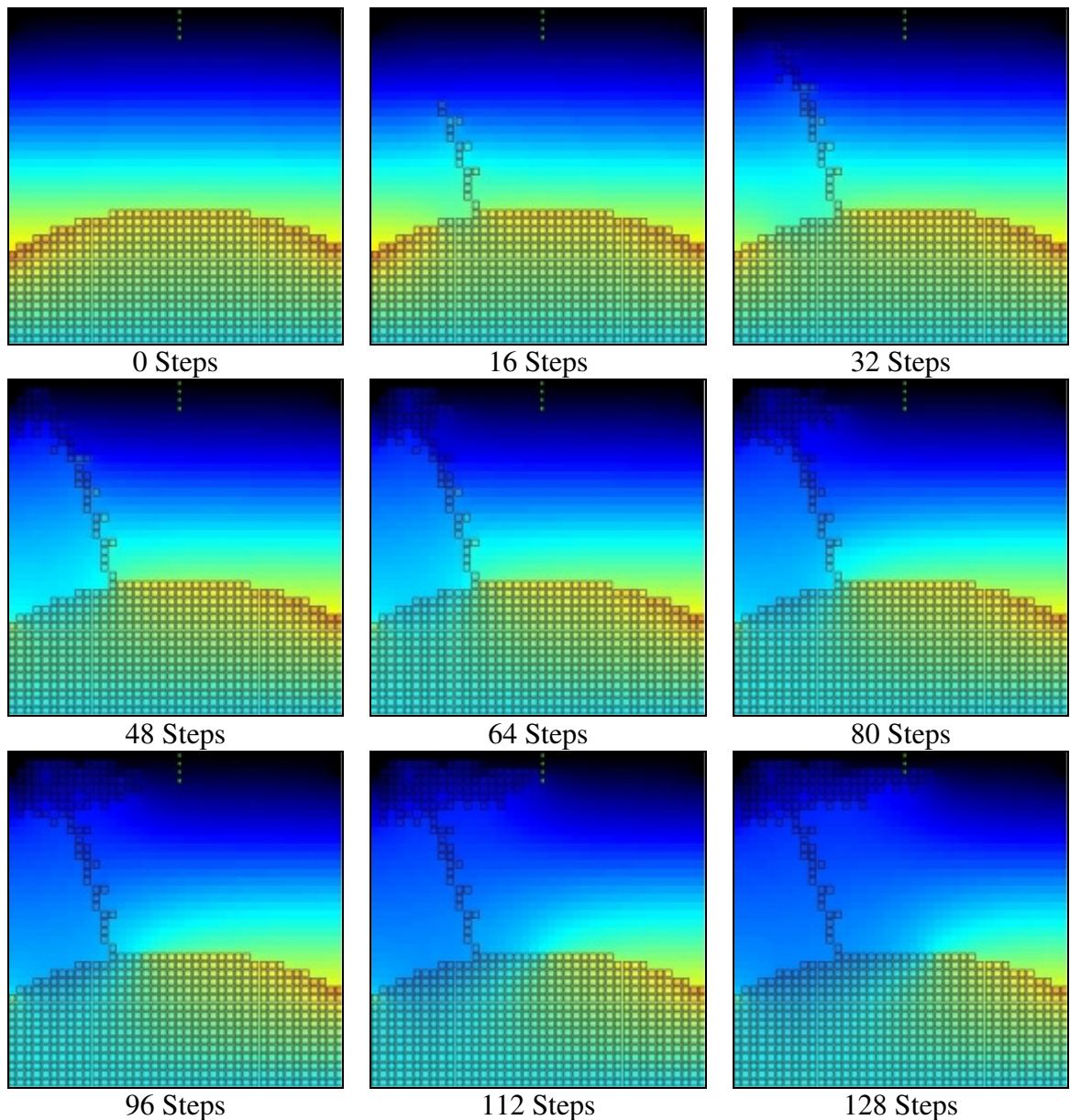
Figure 4.4.3 Example of 1st test: Development of best system found in the interactive runs for one rule. This test has the sun shinning directly down. Note how the structure formed acts as a parasol creating shadow on the entire left side. This has a cooling effect that leaves most of the environment blue or cyan. Cyan represents the target temperature of 50 degree units. A key difference between this and the other two tests is that because the sun is shining straight down as soon as the structure climbs up it cools immediately. This happened because as the structure grows taller it is getting closer to the heat sink but not incurring any extra heat because no sunrays will hit the side of the tower. This effect makes the tower a purely cooling element making the cold seeking agent hang around the top without expanding to the side until it saturates the area.

0 Steps	16 Steps	32 Steps
48 Steps	64 Steps	80 Steps
96 Steps	112 Steps	128 Steps
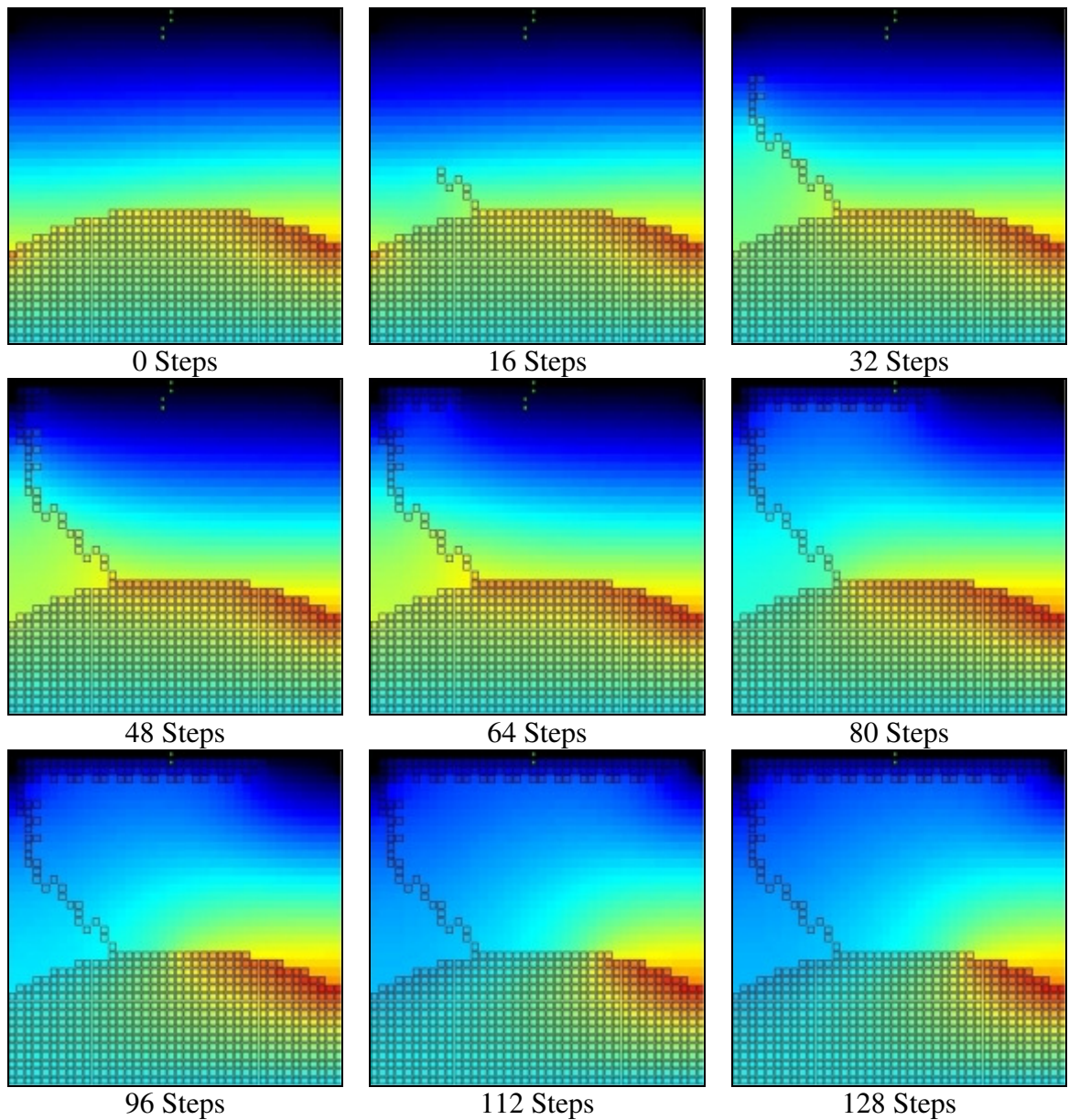
Figure 4.4.4 Example of 2nd test: Development of same system described for previous figure but for test condition number two in which the sunrays are angled at a slope of two. An interesting effect caused by the sun angle is that the growing structure catches more heat and initially heats up the left side of the environment. This shift in gradient seems to be the reason of why the agent moves across the top of the environment more quickly creating a longer roof which consequently is compensating for the angle of the sun.

0 Steps	16 Steps	32 Steps
48 Steps	64 Steps	80 Steps
96 Steps	112 Steps	128 Steps
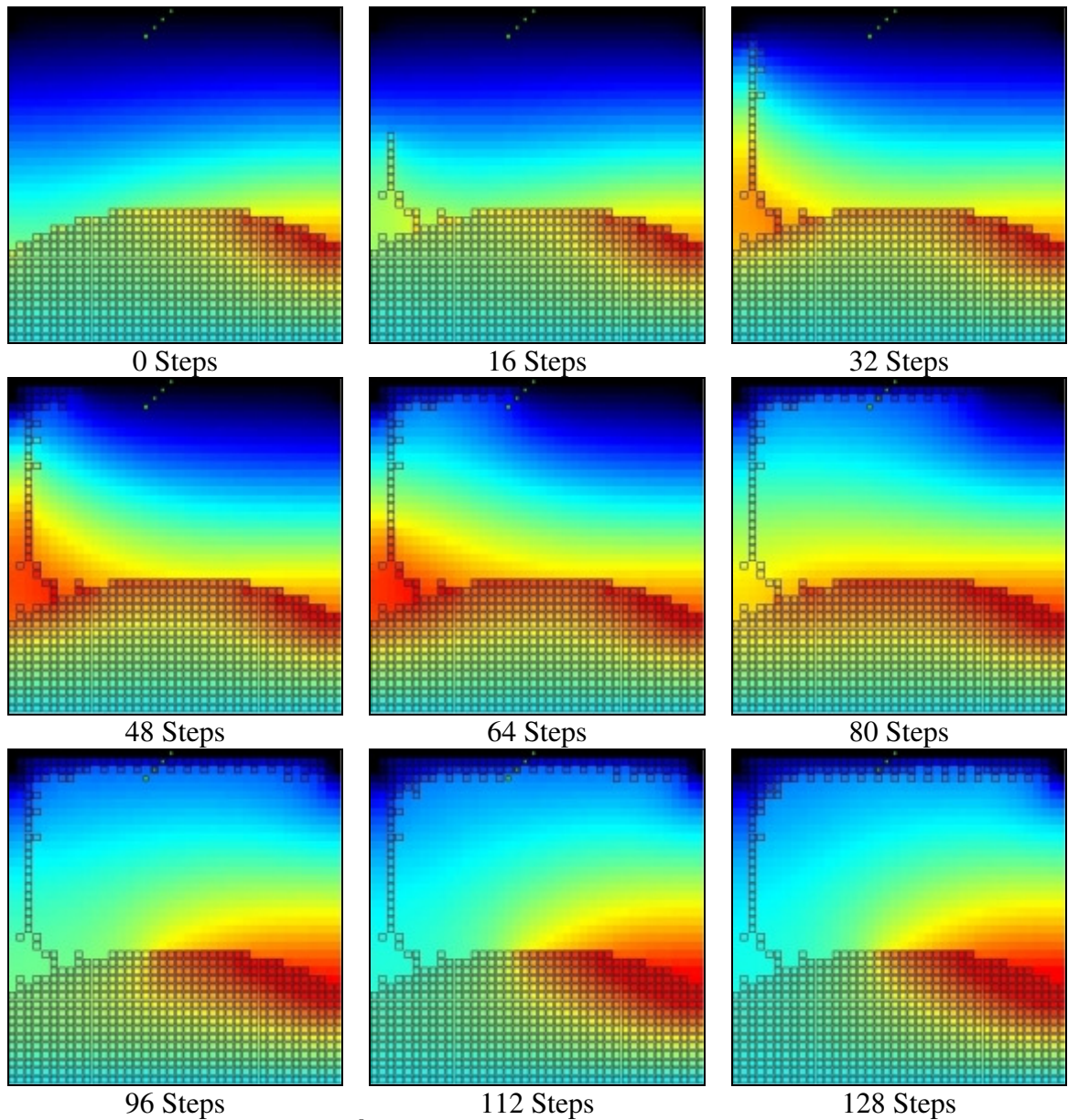
Figure 4.4.5 Example of 3rd test: Development of same system as previous two figures but for test condition number three in which the sunrays are angled at a slope of one or 45 degrees. The same effect of heating the left-side is created by the sun but hotter. The agent takes the approach of going up the side and then across. In this case the agent does it much faster and covers the whole top with steps to spare. At this point the agent would have probably kept going to the right shadowing more area but it hit the edge of the environment. A good direction to go would have been down creating more shadow to cool of the really hot stop of the right side. However no agent was ever able to do this and it is probable that the representation was not rich enough to allow such behavior as it would mean to seek out the warm zone. A switch in strategy would need to be triggered and the agent cannot do this.

CHAPTER 5

RESULTS

The main experimental results of this thesis involve the comparison between systems with a reactive level of system-environment interaction and systems with an interactive level of system-environment interaction. Our hypothesis states that interactive systems and reactive systems are able to achieve the same level of robustness to environment variation. And the question stated by this hypothesis is whether or not the increased system-environment interaction really allows the design of systems with better performance and more robustness to environmental variables?

The results shown in this chapter include graphs of fitness for evolutionary runs, figures of evolved genotypes, and figure of evolved phenotypes.

5.1 Reactive vs. Interactive

The following figures directly address the hypothesis. They show a comparison between runs done with systems that were similar in every way except for having different levels of system-environment interaction.

The experimental data disproves the hypothesis. Systems using dynamic environmental feedback showed an increase in fitness performance and system robustness in building functional structures under different environmental conditions. As mentioned before, a full fitness evaluation in these experiments actually involves three calls of the fitness function because the systems are tested three times under different environments. Therefore the fitness given to an individual is really the average fitness of three tests, which rewards generalists over specialists. A break down

of the three fitness values was also plotted to see how these systems where performing in each test, see Figures 5.1.3 and 5.1.4. The term generalist is being used here for the systems that are evolved using the three different tests. This means that these systems have to optimize for three different conditions at the same time, compromising performance in each individual condition in order to achieve a higher global fitness. The breakdown plots show the fitness trends for each individual test over the course of evolution. The behavior of all the different kinds of test showed some consistent trends that raised more questions.

It is interesting to note that these figures, including those in section 5.2, where made and seen by the author in about the same other as they are presented here.
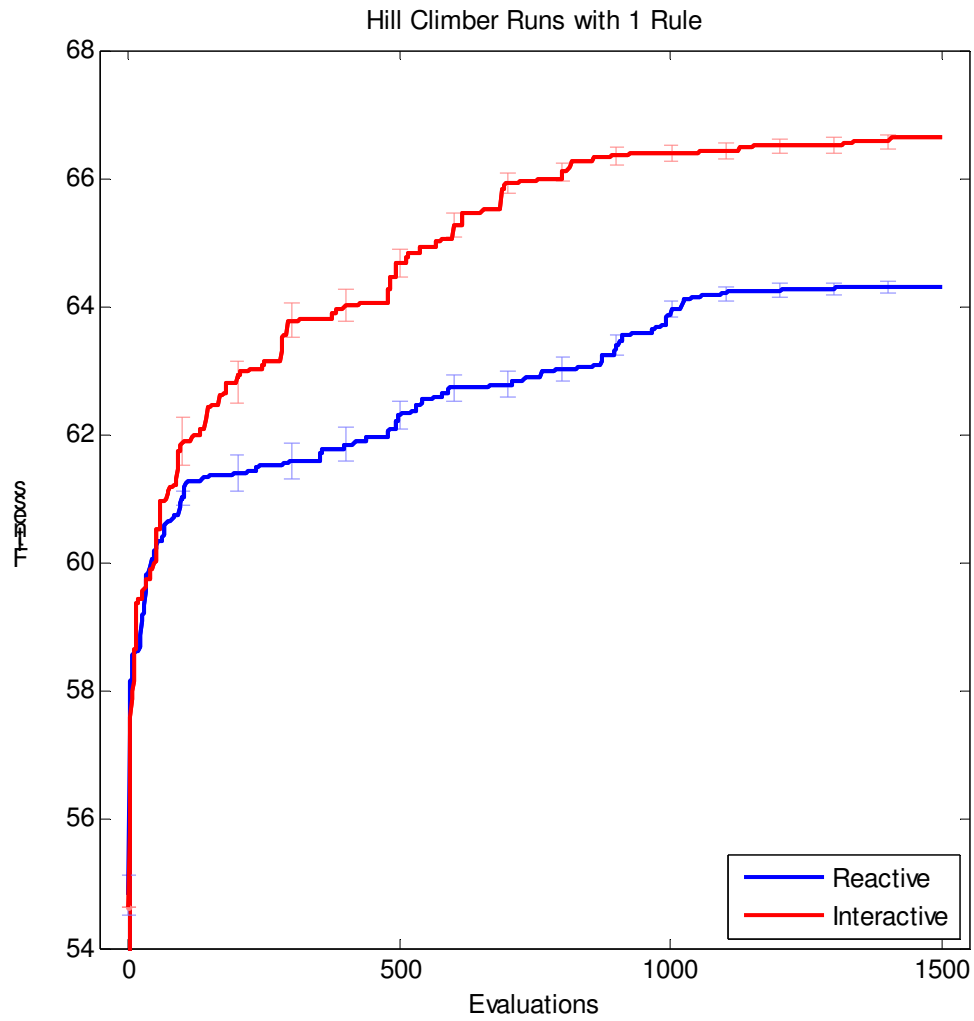
Figure 5.1.1 Averaged from 16 runs and including error bars. Systems with an Interactive Level of system-environment interaction are indeed able to achieve a higher fitness performance when comparing systems that use just one rule. T-test significance = 0.0269
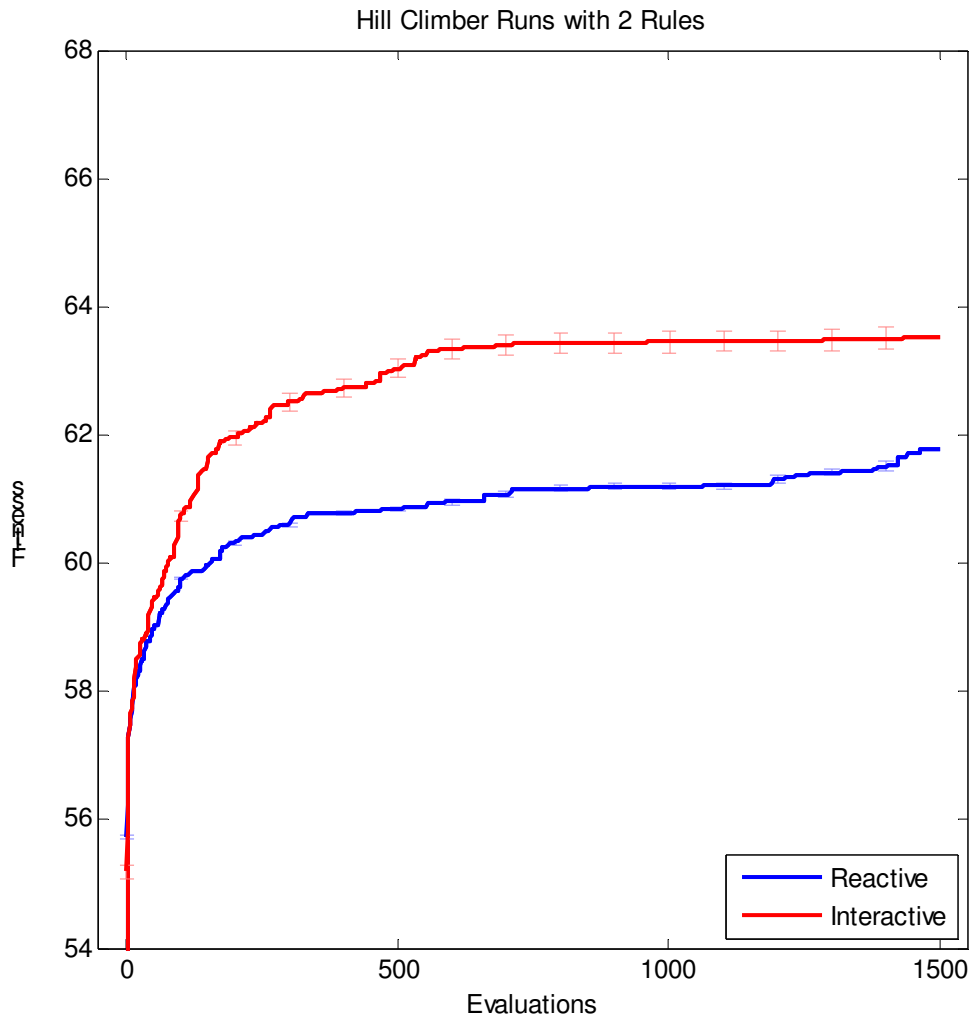
Figure 5.1.2 Averaged from 16 runs and including error bars. When comparing systems that use two rules, systems with an Interactive Level of system-environment interaction also show an increase in fitness performance. T-test significance = 0.2352

Once the breakdown of the fitness values was plotted it became apparent that the fitness in test one was being consistently sacrificed in other to increase the fitness in tests two and three. Such consistent behavior should have an explanation and a closer look needed to be taken. This prompted new experiments to be run. These new experiments are discussed in the next section.
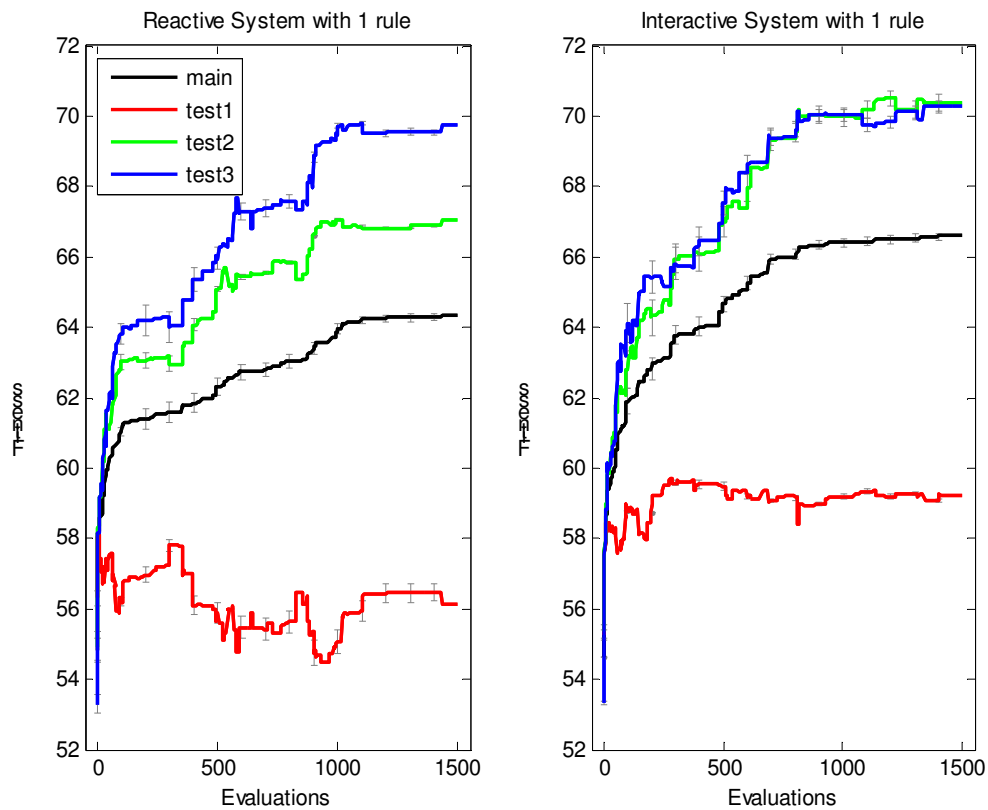
Figure 5.1.3 Breakdown of fitness values in evolved systems with one rule. Note how results in test one were sacrificed in other to increase fitness in tests two and three. The main line is the average fitness of the three tests.

It was strange to find out that in every case systems using one rule out performed systems using two or more rules. It was expected that more rules would allow better systems to evolve. It is possible that the case for this was the execution style chosen for multiple rules, explained in section 3.2. Even with more evaluations, the more rules a system was given, the lower it would top out in fitness. It is also possible that the environment was not complex enough to give an advantage to more complex systems. If this was the case then having more complexity would just create a disadvantage to the system as the evolutionary algorithm would have a harder time evolving the increased amount of parameters.
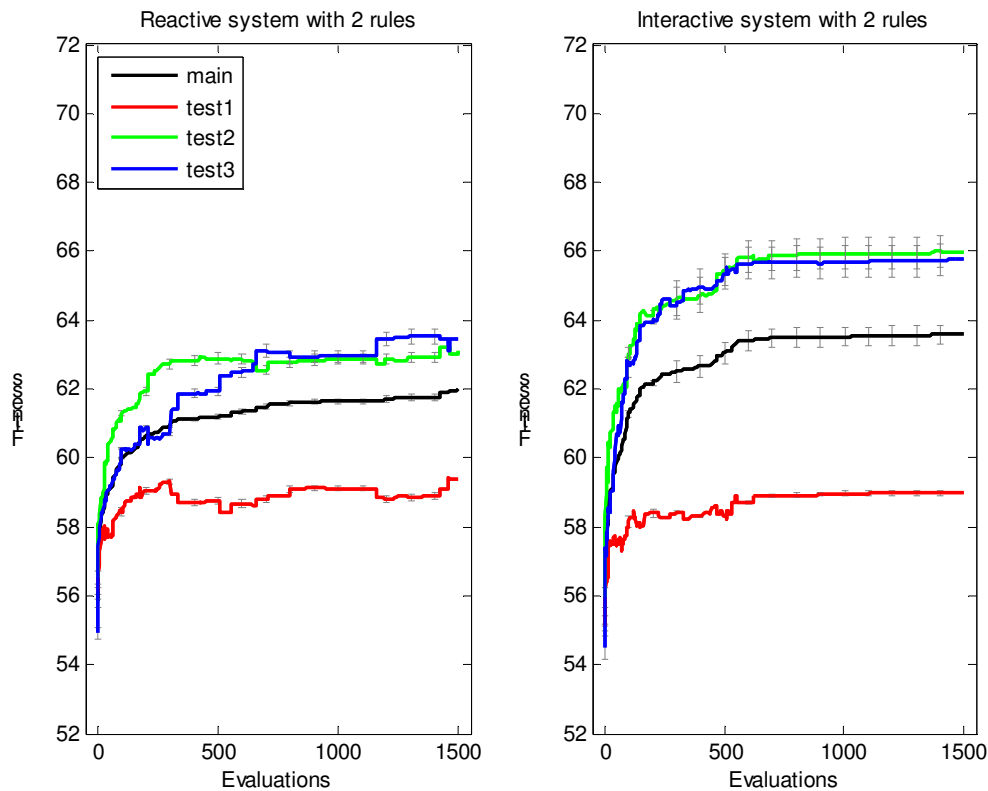
Figure 5.1.4 Breakdown of fitness values in evolved systems with two rules. Again fitness of test one is sacrificed in other to increase fitness in tests two and three. The main line is the average fitness of the three tests.

5.2 Generalists and Specialists

This new set of experiments evolved systems that will be referred to as the specialists. These systems were evolved using the same hill climber algorithm as the generalists, original runs. Definitions for these are as follows.

Generalists Systems are evolved to simultaneously optimize for three different environmental conditions. **Specialists Systems** are evolved only on one environmental condition. Consequently the specialists systems were independently run on the same three conditions as the generalist systems. The following plots again show the results seen in section 5.1 for the breakdown of fitness by test in the evolved systems but

sorted by test and not sorted by run. For Figures 5.2.1 and 5.2.2 the legend abbreviations mean the following:

Re. Spec. – Reactive Specialists

Int. Spec. – Interactive Specialists

Re. Gen. – Reactive Generalists

Int. Gen. – Interactive Generalists

Generalists vs. Specialists for systems with 1 rule
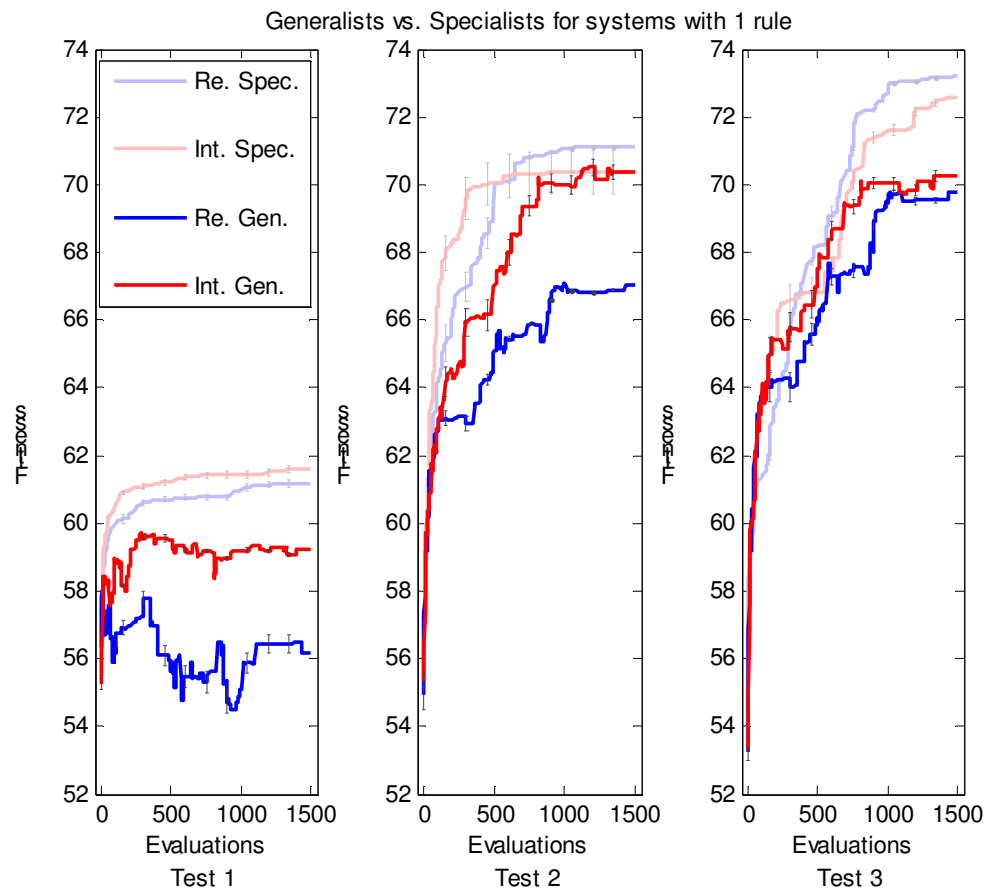
Figure 5.2.1 each line represents data from 16 evolutions runs. The generalist data was all taken from the same 16 runs but the specialist data is from three different sets of 16 evolutionary runs.

The new test data, specialists systems, is represented as the lighter graphs. Since these systems where evolved specifically for each test it could be assumed that they symbolize the top performance we could expect from the generalist system. In Figure

39

5.2.1, which is for systems using one rule and shows results for both Reactive and Interactive systems, we see this as mostly true. In only one case does the generalist system manage to reach the level of one of the specialists and that case was for the Interactive system in test number 2. However note the error bars on the Interactive Specialist runs for test number 2. These errors are quite wide suggesting that in many case the runs would reach a much higher fitness and in other cases get stuck at much lower fitness.

A very important observation from Figure 5.2.1 is that although the hypothesis is supported by previous data plots, the comparison of the performance in different tests side by side of the generalists interactive and generalist reactive systems shows furthermore that the interactive system are able to consistently create better structure in all three different environmental conditions. The hypothesis states that interactive systems can display more robustness and Figure 5.2.1 shows this quite clearly. Furthermore in the specialist cases, which do not account for any robustness, the difference in performance between interactive systems and reactive systems is much less noticeable and in fact favors the reactive systems in two out of three tests.

When the hypothesis was formulated it was thought that the interactive systems would always out perform reactive systems. But it seems that although interactive systems are indeed more adaptive and show more robust behavior. When it comes to specializing for a single environmental condition is it easier to evolve reactive systems than interactive ones. Even though reactive systems can win this battle it is by a small margin and not in every case.

In Figure 5.2.2 the same results are shown but for systems with two rules. These results are actually quite erratic and do not show many consistent trends. The most marked feature from these plots concerns the generalist data. This feature is the level

of smoothness compared with the one rule plots. This supports a previous thought that the two rule systems are less evolvable than the one rule systems.

This figure does show some very interesting oddities, such as the generalist system out performing the specialist in tests 2 and 3. How is this possible? Maybe the specialist systems found it too easy to get stuck at a local maximum when using 2 rules. This maybe was not the case for the generalist system because they had more information to work with or to nudge them out of a local maximum.
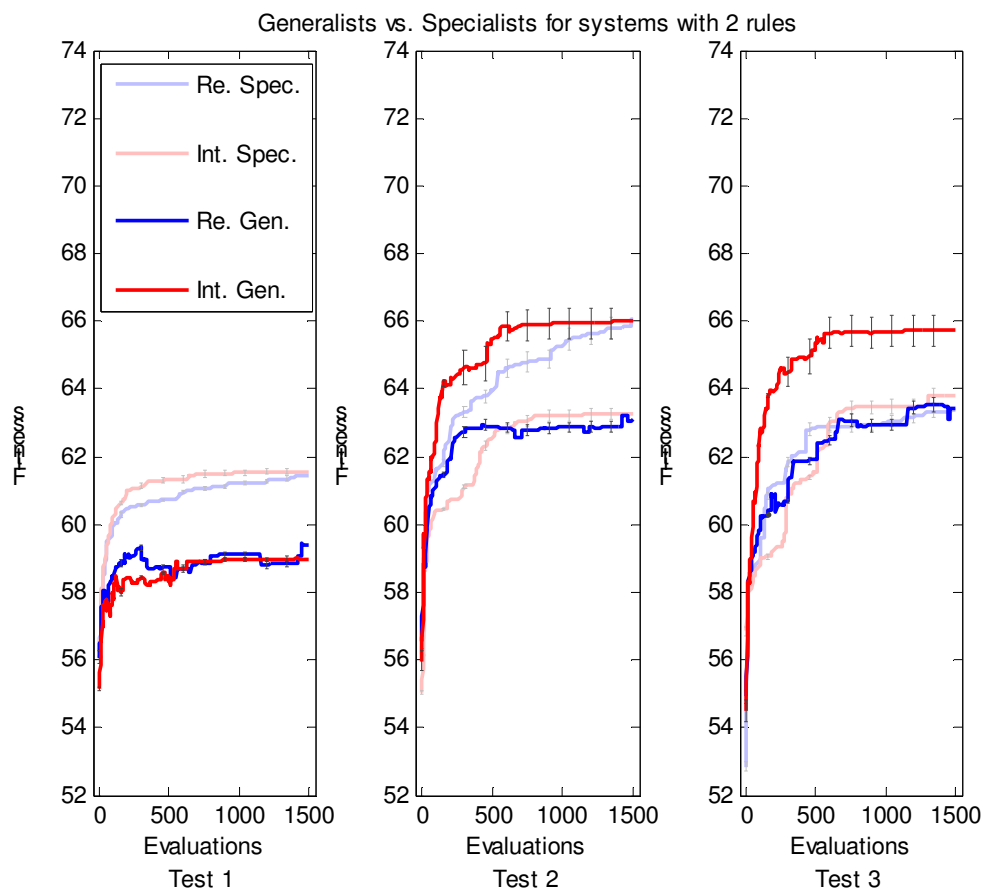
Figure 5.2.2 each line represents data from 16 evolutions runs. The generalist data was all taken from the same 16 runs but the specialist data is from three different sets of 16 evolutionary runs.

5.3   *Genotypes*

A look was taken at the genotypes in order to explore trends in the evolved rules. For the single rule genome plots each dot in the plot contains six types of data within it, the five genes plus the fitness for that genome. Table 5.3 contains a list of the information included in the genotype plots and how they are represented.

Trends show a definite link between the sensor genes and the fitness. All the high fitness agents are located at low temperature meaning the agent would be seeking out the cool. The density does not seem to be as strong a factor as temperature but the fitness still favors densities below 50, which indicates that the agent would not favor building in dense areas and would tend to favor branching out away from the solid mass.

The following list describes the elements represented in Figures 5.3.1 thru 5.3.4. These figures show the final genome from the four different runs.

Mean Temperature -   X-Axis

Mean Density -   Y-Axis

Action -   Cross = 1, Circle = 0

Angle -   Line that extends out from dot

Direction -   Notch at end of angle line

Fitness -   Red = High, Blue = Low

Final Genomes for Reactive runs with 1 Rules
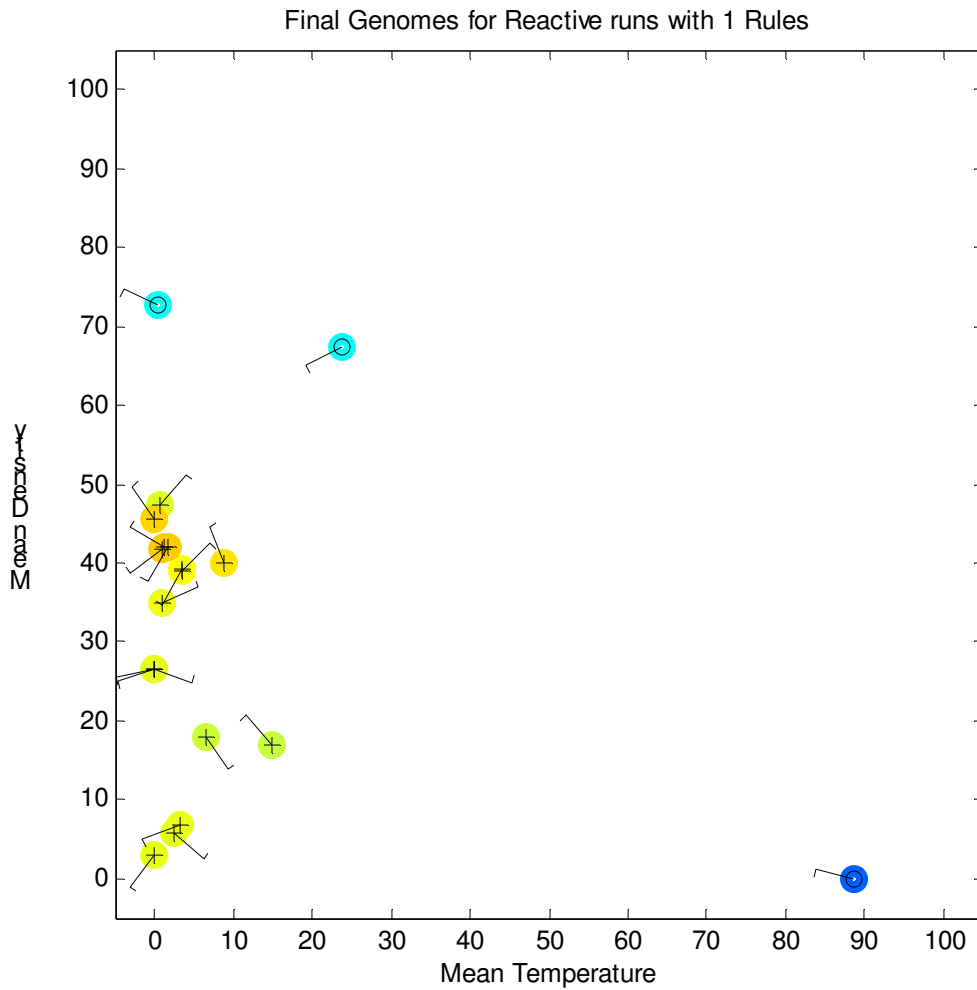
Figure 5.3.1 Evolved genomes of reactive Systems with 1 rule

It is interesting to note that in both the reactive and interactive systems with one rule the evolution concentrated in the low temperature and medium to low areas of density, yet the interactive system achieved high fitness. I both cases all the high fitness individuals are "builders", meaning that their action genes is one and all they can do is build solid blocks. The few runs that got stuck in lower fitness areas are all "diggers" at higher values of temperature and density. This means that they were probably stuck at some local peak that required several simultaneous mutations in the right direction in order to jump out of this "bad solution".
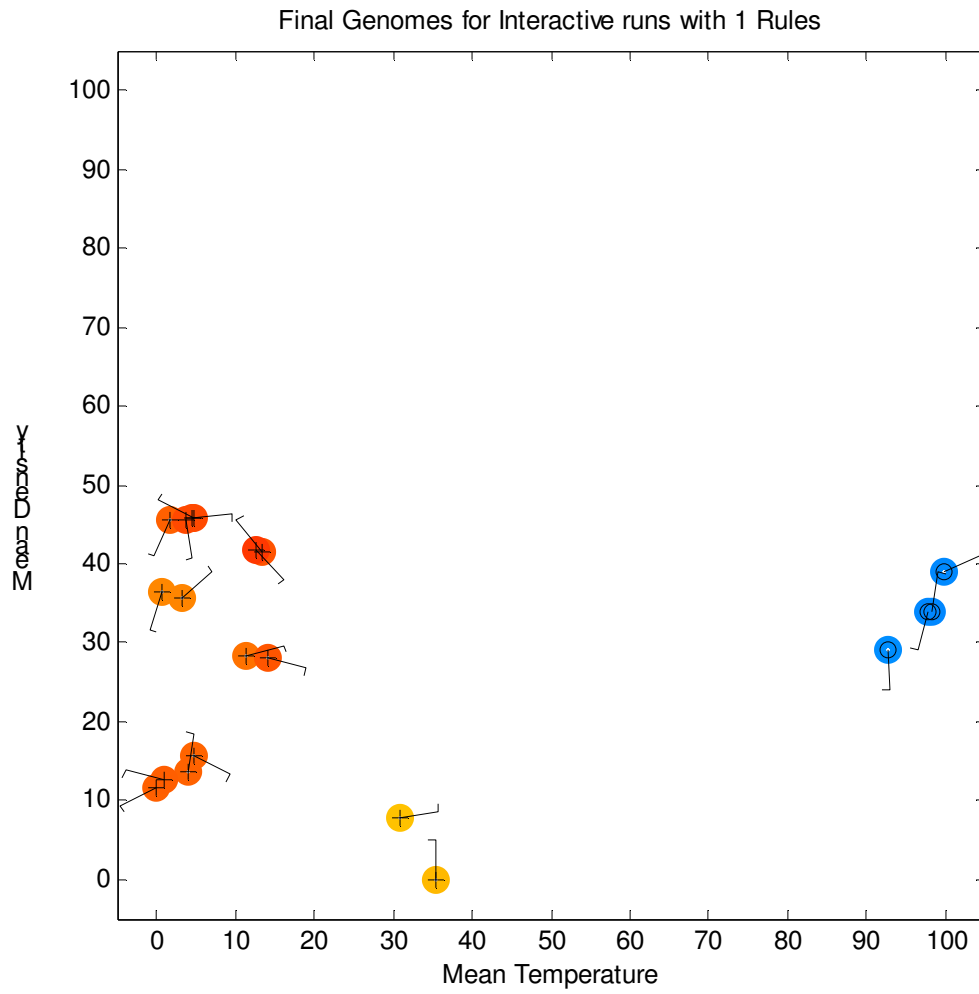
Figure 5.3.2 Evolved genomes of interactive systems with 1 rule
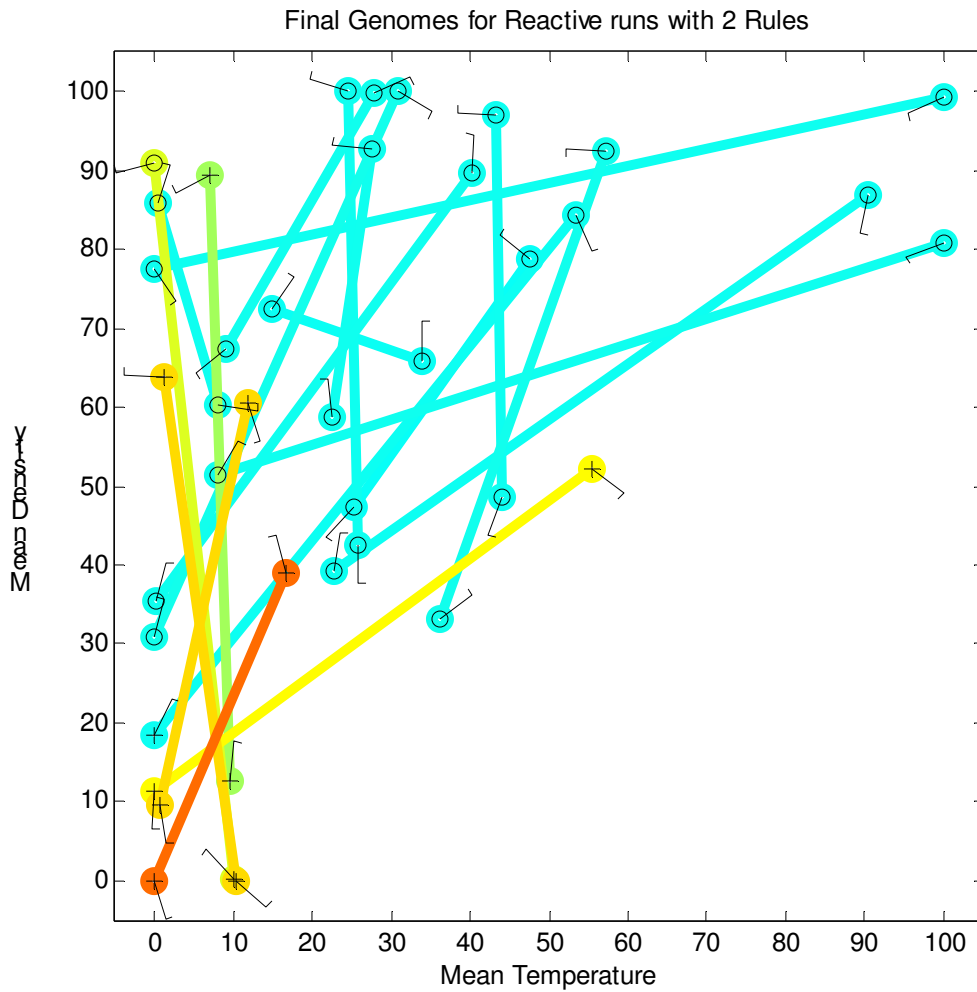
Figure 5.3.3 Evolved genomes of reactive systems with 2 rules

In the case of reactive systems with 2 rules the evolutionary algorithm was not able to find the optimal solutions as consistently as with the one rule systems. It seems that within the representation the high fitness zone it still within the same zone as with the agent with one rules but in the genomes would be to have both rules be near the same zone and be both builders. Again the same conclusion regarding the representation used for systems with multiple rules can be stated. The representation

simply could not take advantage of the extra rules, and in fact having more rules just hurt the evolutionary process.

For the interactive systems that used two rules, there was a noticeable difference regarding the fitness in their results. Somehow the interactive systems were able to better seek out the favorable genomes, although they also managed to have worse fitness failures.
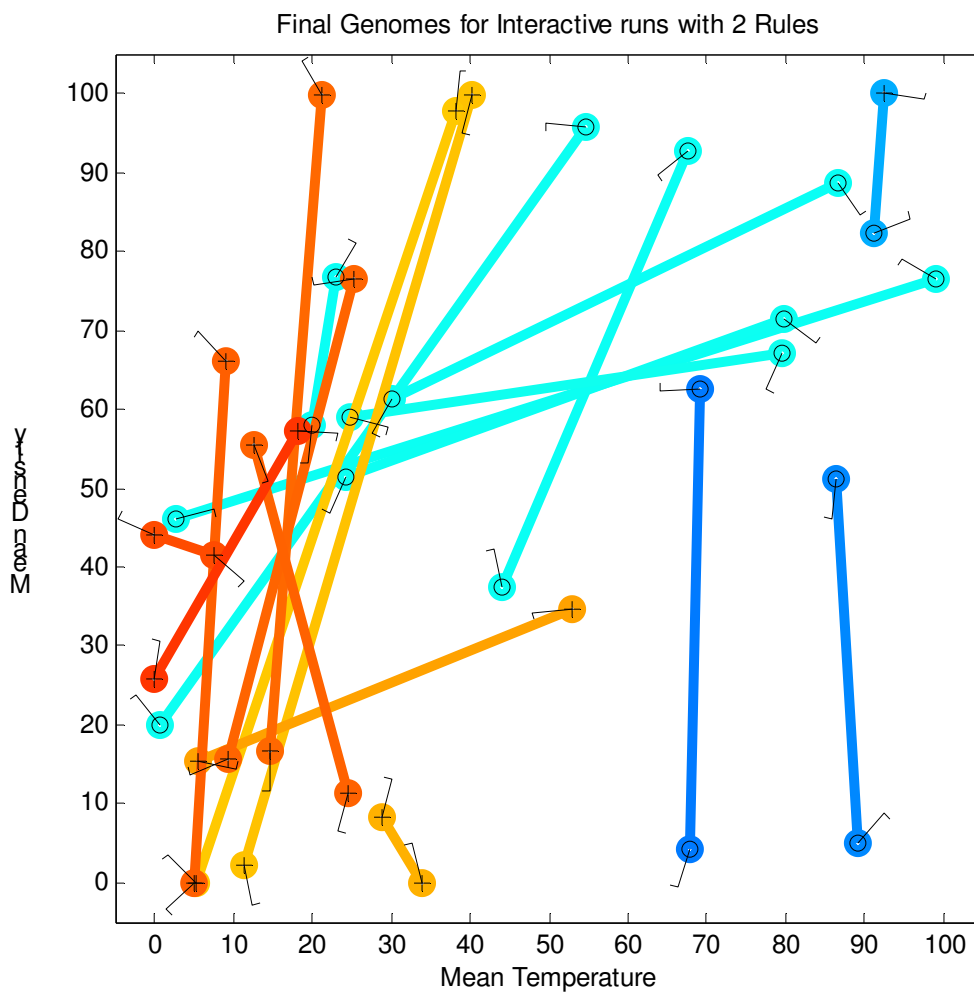
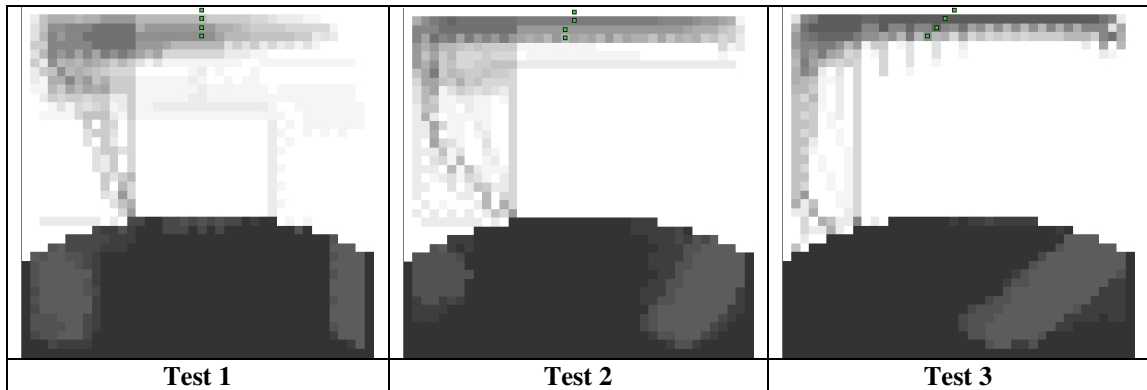**Figure 5.3.4 Evolved genomes of interactive System with 2 rules**

In the two rules systems the reactive level was a lot more consistent at finding medium range fitness while the interactive system would have more runs end with good and bad fitness values while spending much less time at the middle.

5.4 Phenotypes

The Phenotypes in the systems used in these experiments are the resulting structures that emerge from their development, which is guided by their genotypes. From the genotype plots that were discussed in the previous section it is already know that the system that evolved a digging behavior scored lower fitness than the builders. In these figured the final state of all the runs of a type were averaged for each test. The darker areas represent the block which tended to be solid more often at the end of runs.
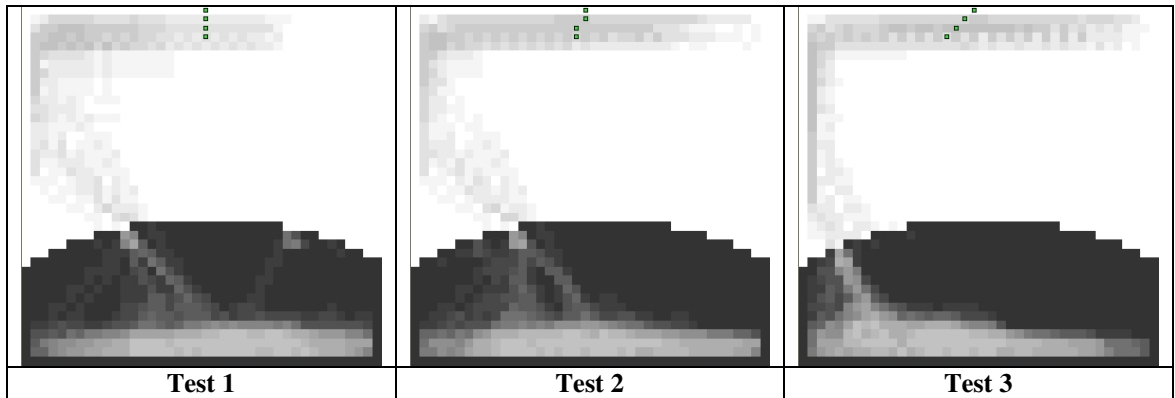

| Test 1 | Test 2 | Test 3 |
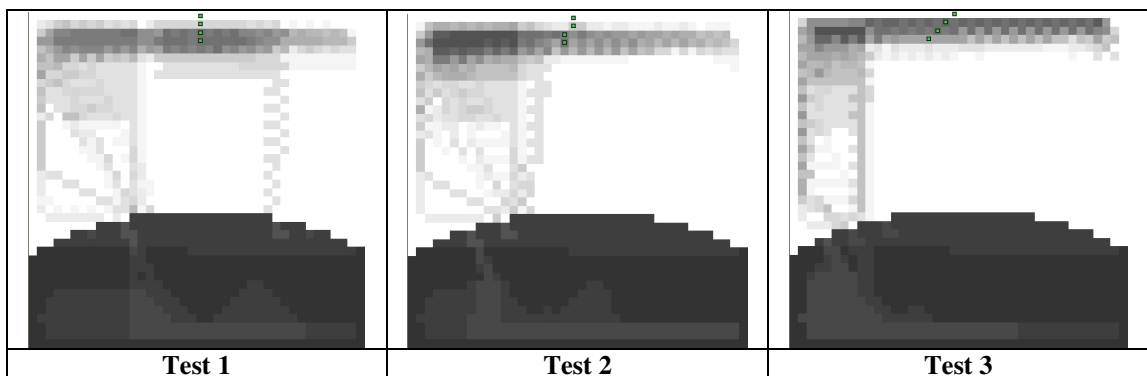
Figure 5.4.1 Evolved phenotypes of reactive systems with 1 rule

| Test 1 | Test 2 | Test 3 |

Figure 5.4.2 Evolved phenotypes of interactive systems with 1 rule

The most notable feature in these figures is that most builder systems evolved into the canopy strategy. This strategy consists of building a relatively thin column in order to get to the top of the environment and then build sideways to create shade. This strategy has a cooling effect as the sun stopping cells are very close to the zero degree heat sink so they do not warm up.

It is quite probable that the problem present was too simple or the environment not rich enough for the different systems to evolve different strategies that where just as good. It seems that for the conditions presented and the rules used the canopy strategy was the overall winner no matter what system was evolving. In chapter 6 this issue is explored further and experiments are shown where the interactive and reactive system use completely different strategies yet achieve comparable fitness values. Also experiments are shown where there is virtually no difference between the interactive and reactive results. The variation in these experiments is only in the initial conditions presented to the individual. The physics remain the same.

| Test 1 | Test 2 | Test 3 |

Figure 5.4.3 Evolved phenotypes of reactive systems with 2 rules

| Test 1 | Test 2 | Test 3 |
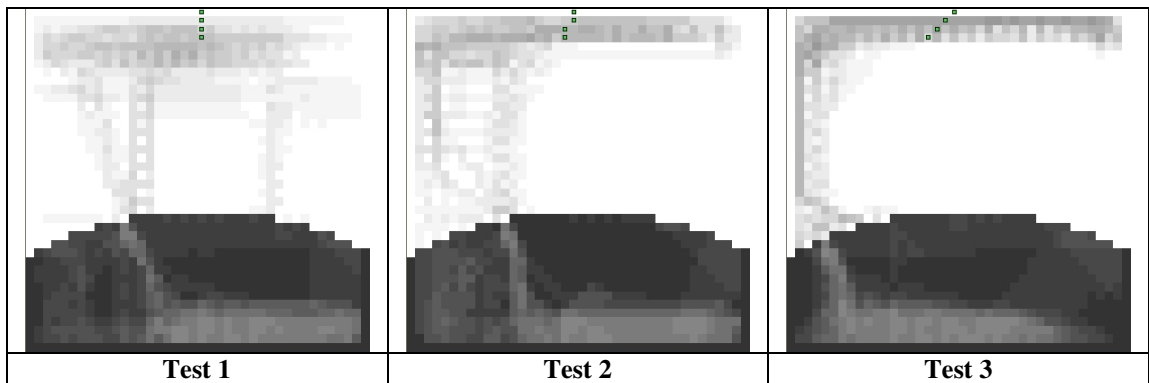
Figure 5.4.4 Evolved phenotypes of interactive systems with 2 rules

The systems run of all types that used digging agents do not seem to have any overall pattern, but diggers evolved within the same type of run do tend to have the same pattern. For example the interactive system with one rule only evolved diggers that would make holes at both sides of the hill. While the system with 2 rules would tend to just dig down in no particular way and then expand the hole sideways at the bottom, like a reverse canopy strategy.

CHAPTER 6

DIFFERENT SCENARIOS

This chapter showcases six further experiments done with three different initial conditions and for both reactive and interactive system. These experiment where done to find out whether or not our two systems would behave differently from each other under these new arbitrary initial conditions. This study was prompted from the phenotype results in our main set of experiments.

6.1 The Circle Scenario

The starting condition for this scenario was a circle of solid cells with radius of 8-cells just below the center of the environment surrounded by empty cells.
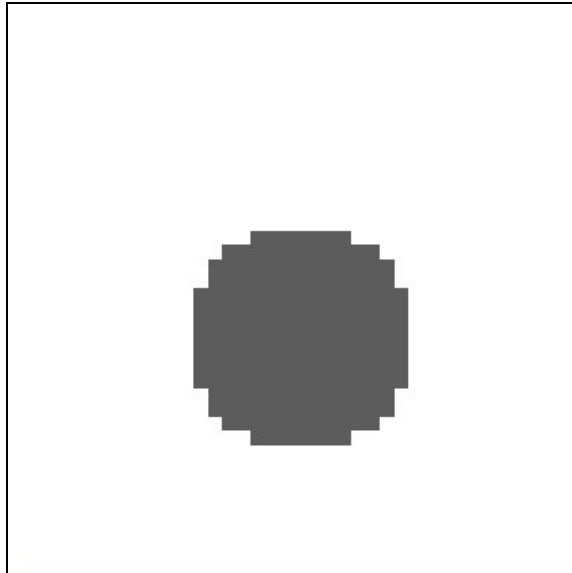

Figure 6.1.1 Diagram showing the starting conditions for the circle scenario. Grey shows solid cells and white show empty cells.
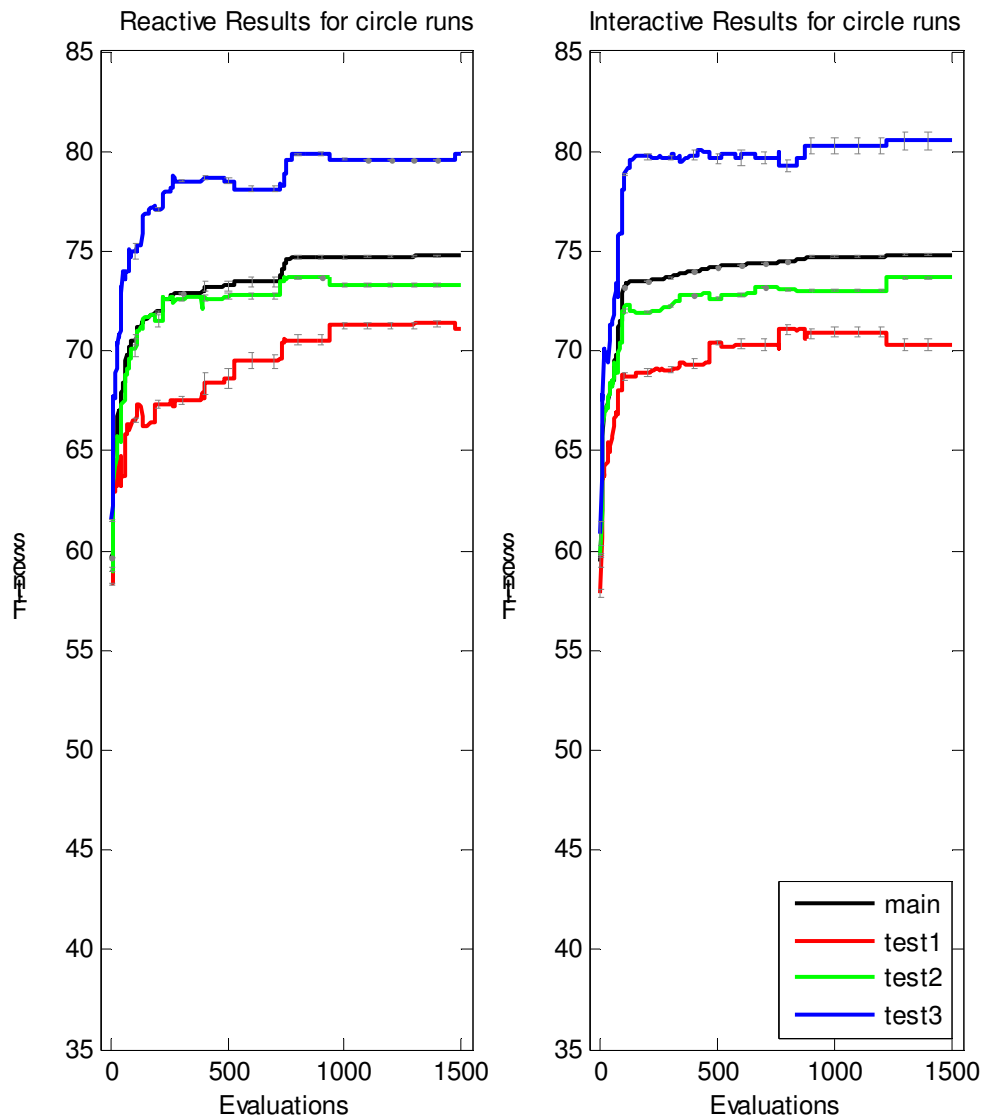
Figure 6.1.2 Circle Fitness Results: Plots of the reactive and interactive run for the circle scenario. Results are almost identical between reactive and interactive. Yet both have overall high fitness values compared with every other scenario run.

It seems that this scenario was able to evolve into much higher fitness values than all the other scenarios, including the main experiments. Looking at the results it is quite possible that the biggest factor in this was the empty space in the bottom of the
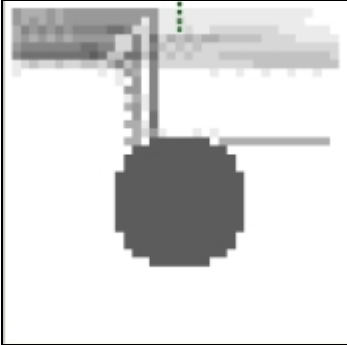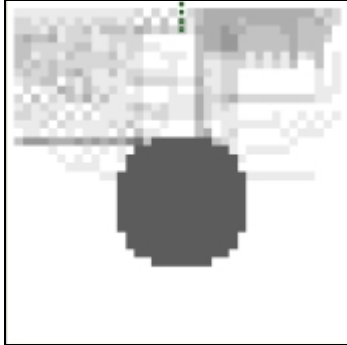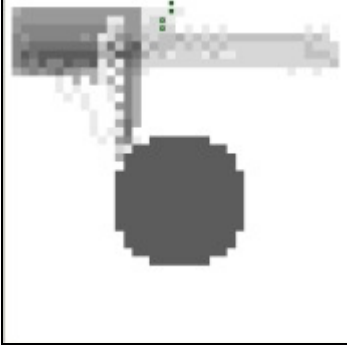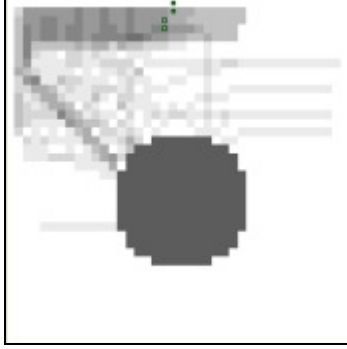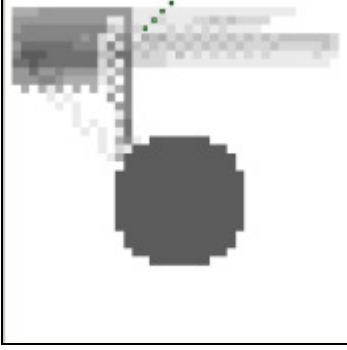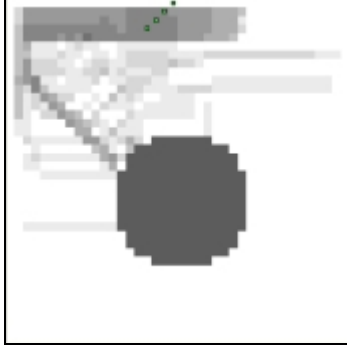
environment region. This area remained mostly undisturbed by the evolved phenotypes, see Figure 6.1.4, and its temperature was always very close to 50 degrees thanks to the bottom row always being at 50 degrees because of the constraints.
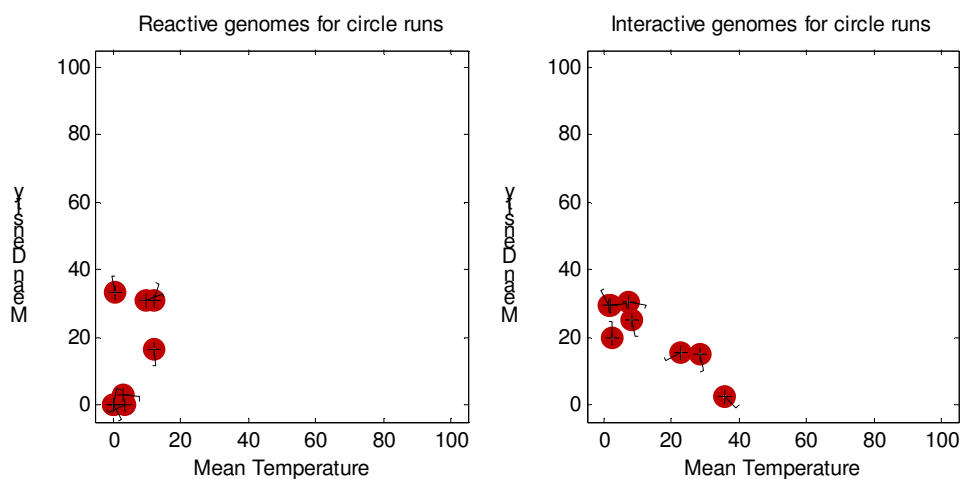
Figure 6.1.3 Circle Final Genomes: Plots showing the genome distribution for the circle scenario. Just as the fitness plots, both reactive and interactive runs show near identical outcomes.

These final genomes indicates that, for the circle scenario, the fitness landscape was comparatively more evolvable than all the other scenarios seeing as how all the runs ended in similar conditions with low temperature, low density, and all where builders(no diggers). Also all scored high fitness values (all dark red). This is a possibly smoother landscape with one dominant peak.

With regards to the phenotypes the most notable difference is that the interactive runs tend to have a noisier output with regards to final structures, probably due to the dynamic effects on the temperature gradients during development whereas the reactive individual dealt with smoother and static temperature gradients.

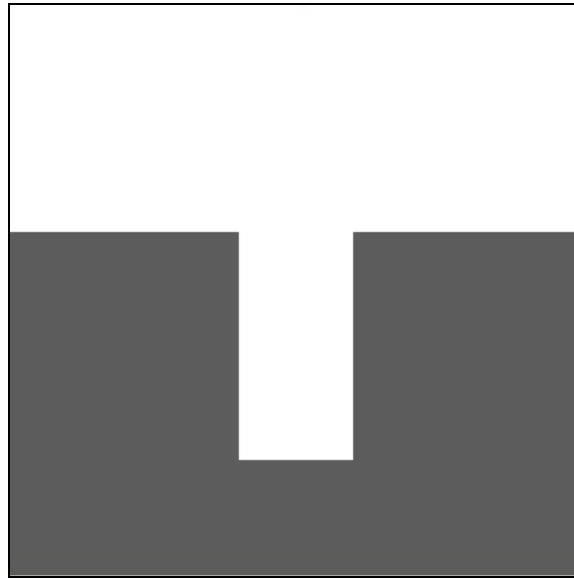| Test	Reactive	Interactive
1		
2		
3		

Figure 6.1.4 Circle Phenotypes: Both reactive and interactive runs found the same basic solution of trying to create a partial roof structure on the upper left side of the environment.
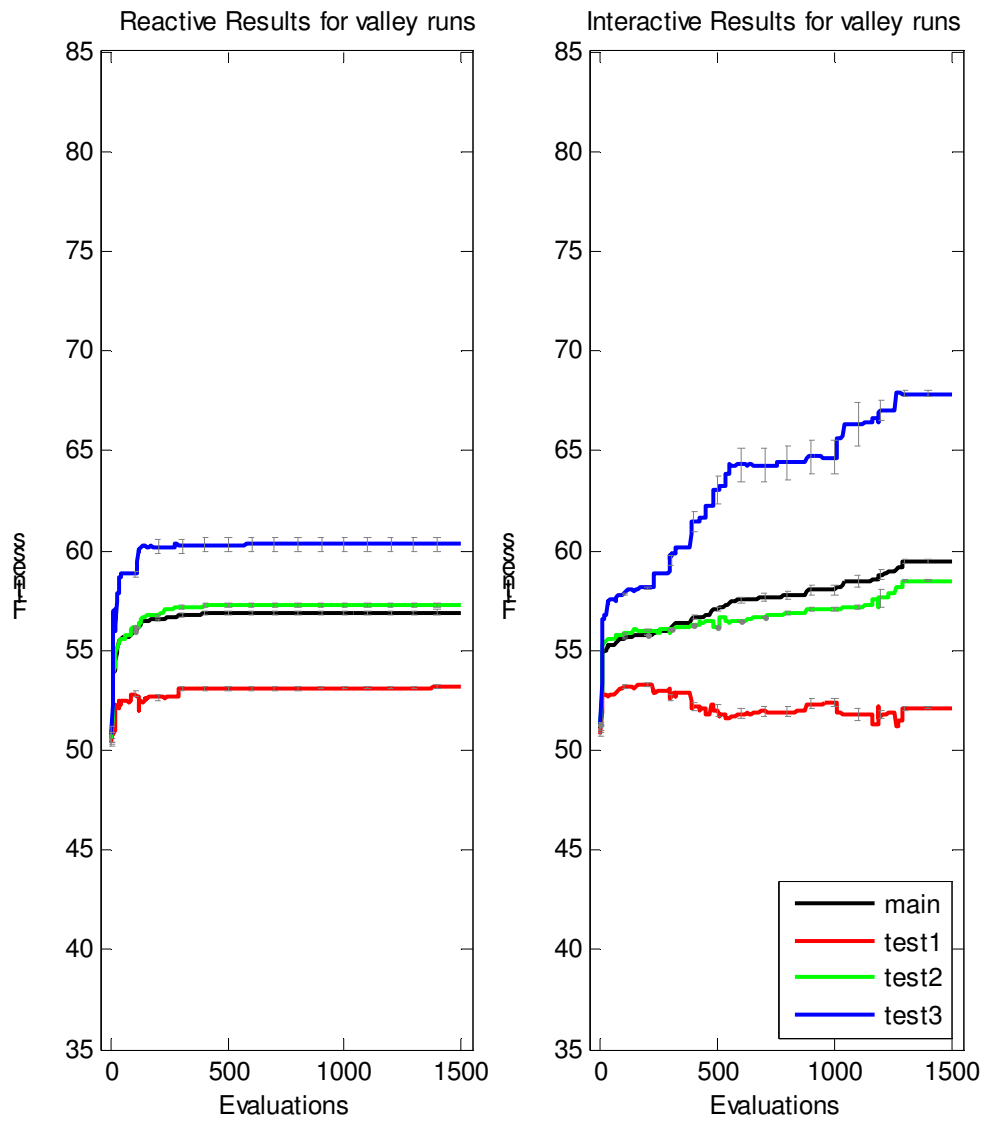
6.2 The Valley Scenario

The starting condition for this scenario was a large solid block covering the lower 60% of the environment area with a valley cut out of it at the center.

Figure 6.2.1 Diagram showing the starting conditions for the valley scenario. Grey shows solid cells and white show empty cells.

In the valley scenario the interactive system was able to evolve into overall better results, Figure 6.2.2, although these interactive runs tended to specialize for tests number 2 and 3, whereas the reactive runs where not able to evolve much at all, mostly settling after about 300 evaluations. Curiously enough the interactive runs sacrificed test number 1 so much that it underperformed the reactive runs for that test.

Figure 6.2.2 Valley Fitness Results: Plots of the reactive and interactive run for the valley scenario.
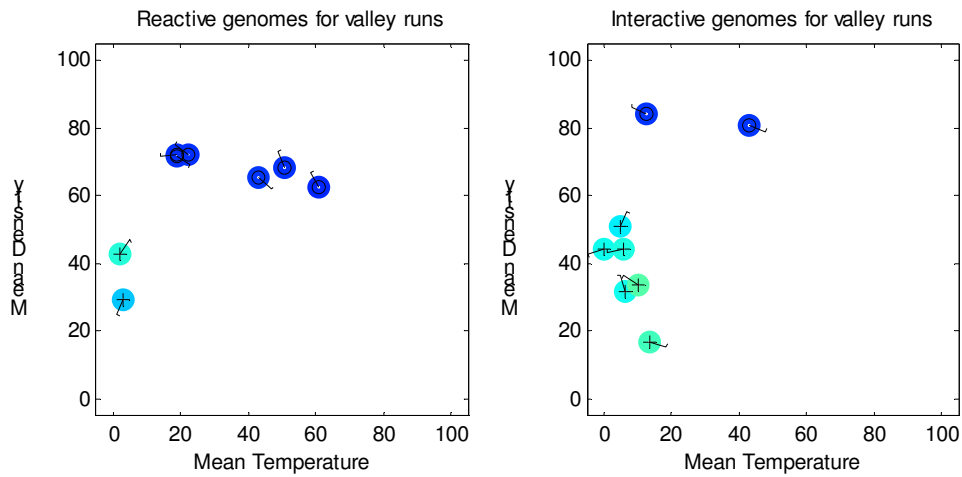
Figure 6.2.3 Valley Final Genomes: Plots showing the genome distribution for the valley scenario.

These genomes plots seem to indicate that the reactive system where able to evolve their best system only a couple of times while the other six runs where pretty much misses. The opposite was the case of the interactive runs, where the evolution got 6 hits and 2 misses. Meaning that most of the time it was able to evolve to its best individuals.

For the valley scenario the interactive system was better able to evolve good systems.

Phenotype wise the results are again similar in the sense that the reactive systems' builders and diggers are similar to the interactive systems' builders and diggers. The difference being that the interactive system evolved builders more often, which incidentally represent the "good" genomes for these runs.
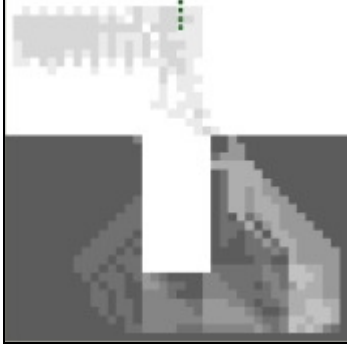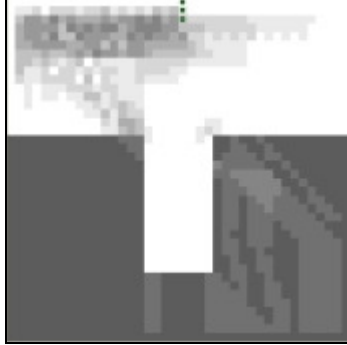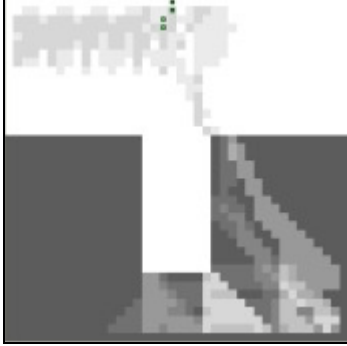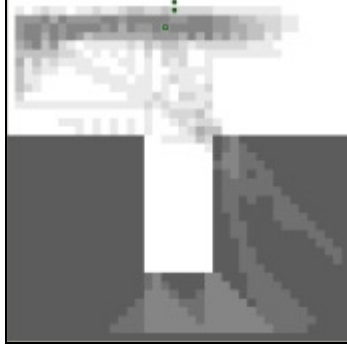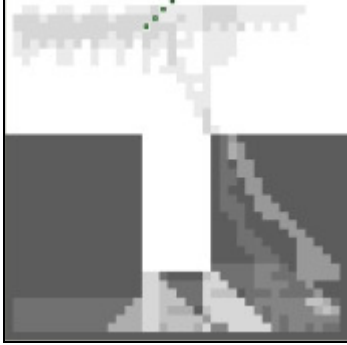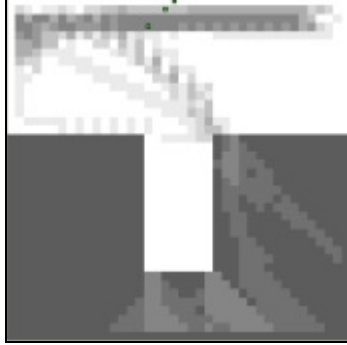
Test	Reactive	Interactive
1		
2		
3		

Figure 6.2.4 Valley Phenotypes

6.3 The Vertical-Bar Scenario

The starting condition for this scenario was the same hill type scenario from the main experiments but with the addition of a high vertical-bar tower on the right side of the hill.
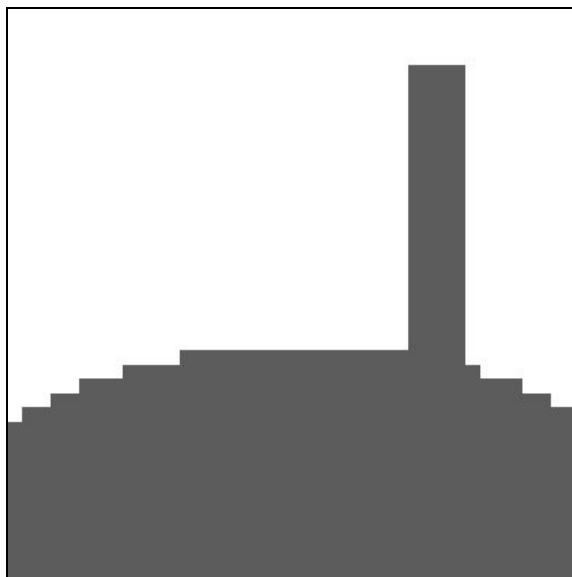
Figure 6.3.1 Diagram showing the starting conditions for the valley scenario. Grey shows solid cells and white show empty cells.

This is the scenario that produced the most interesting results. Both types of runs, reactive and interactive achieved on average about the same overall fitness. The interesting part is that they used completely different strategies. The interactive systems evolved in perhaps the most consistent manner that has been shown in any of the runs presented here. In Figure 6.3.3 we see this as all of the genomes evolved form a small cluster around a mean density of 20 and a mean temperature of 60. Also these are all diggers and well it seems that, as always, no preference regarding angles or direction (4th and 5th genes in a rule) can be observed.
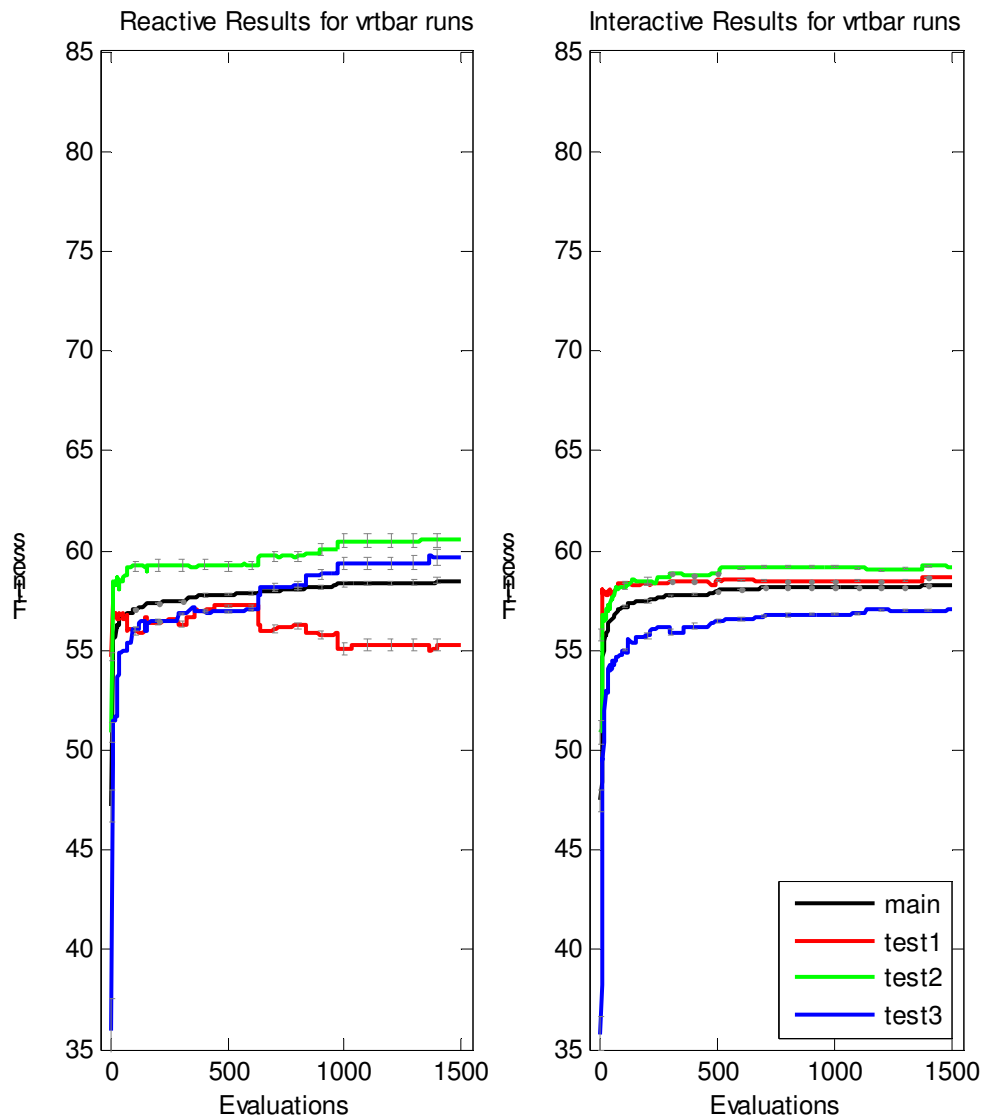
Figure 6.3.2 Vertical-Bar Fitness Results: Plots of the reactive and interactive run for the vertical bar scenario.

The fitness result for both cases was very close with no significant distinction between the "main" fitness plots. Which is curious as the individual test result seem to have no relation in their patterns. It is very rare that we had runs where the test favoring patterns were different. It seems that the case is that the favored test depends mostly

on the scenario used. But these results show that it is also heavily affected by the strategies used. This is a significant result because it breaks a previous pattern that could not be conclusively explained.
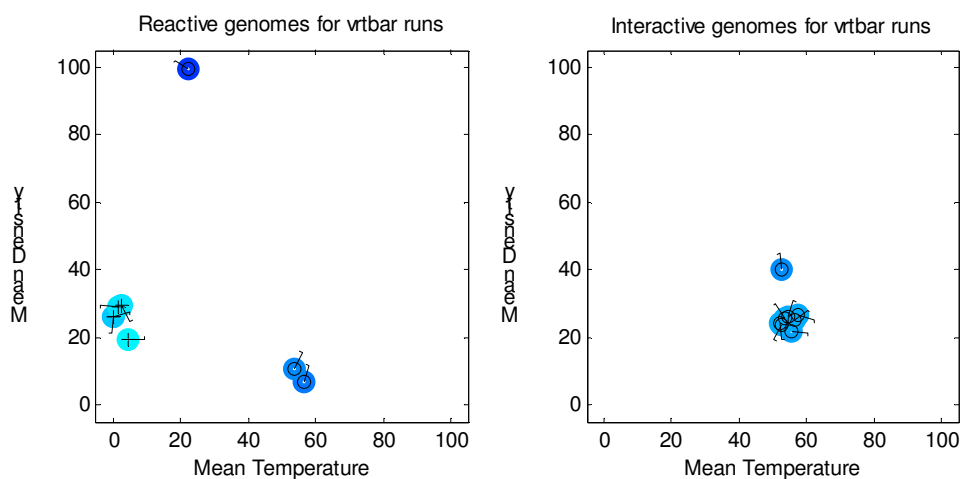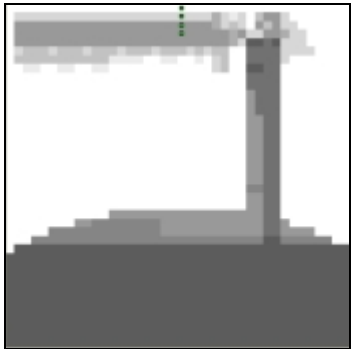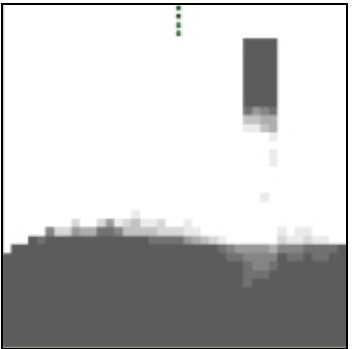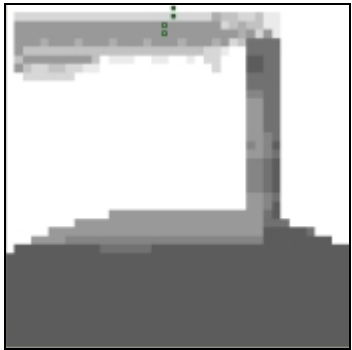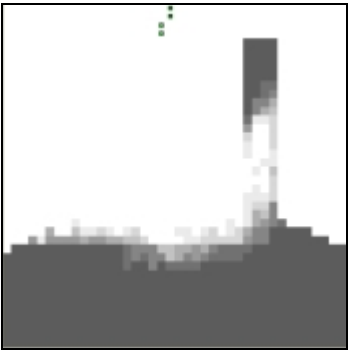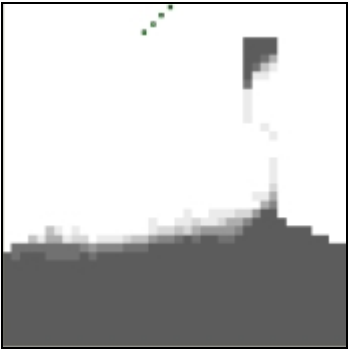
Figure 6.3.3 Vertical-Bar Final Genomes: Plots showing the genome distribution for the vertical bar scenario.

The reactive run seemed to have various local peaks that it settled on, which is quite the opposite from the interactive runs which have one dominant narrow peak. The reactive peaks vary between diggers and builders but in the reactive runs there is one dominant peak only that corresponds to diggers.

In the interactive runs it is noticeable how the digging function shifts its focus with the angle of the sun almost forming a wide hole along the direction of the sun starting from just below the tip of the vertical-bar tower.

The reactive systems however resorted most successfully to the canopy approach but extending from the tower and not the up the left side as with the original experiments shown in the previous chapter.

Test	Reactive	Interactive
1		
2		
3		

Figure 6.3.4 Vertical-Bar Phenotypes: Note this is the only scenario were the reactive and interactive systems consistently pursued different strategies.

It is not clear why this scenario was the only one to produce such results the feature that makes this scenario stand out the most among all the other is that this is the only left-right asymmetrical scenario.

CHAPTER 7

SYSTEM SELF-REPAIR

Perhaps one of the biggest advantages of using Interactive Systems as representation of structural design is that they do not need to stop working, which can lead to different outcomes. One outcome could be an ever growing structure, a design fit for types of structures which need to meet ever growing demands or structures that need to achieve partial functionality before they are finished. This could be the case when trying to establish a functional remote facility that will receive its crew in growing stages. This is not exclusive for remote facilities, but the specific advantages offered make it more attractive in such cases when compared to conventional blueprint methods. Also the word "remote" is being use loosely here and as it can be applied to any case where there is considerable operations establishing cost. Another outcome, which was further explored, is the possibility of a self-repairing system.

7.1 Self-Repair

Consider a rule-based design evolved such that once the desired function is achieved the agents or robots that built the structure could continue in a roaming mode, simply because the feedback from the structure or environment is not prompting build rules, while no spots that trigger build rules are found. In such a case the agent would roam indefinitely, until its power source ran out or a low-power trigger would signal a return to a charging station and then resume roaming as if the construction was not finished because it has no finish trigger. Power source issues aside, the robot-agents rules could be such that they roam indefinitely and, during the

roaming stage, if the agent were to run into an area of the structure that was damaged it could perceive this as simply unfinished and then would go forth and fix the damage before continuing its roaming.

Such automated repair is simply just part of the representation and execution of interactive system design. Of course an designer would probably be tempted to include some sort of extreme damage trigger and sequence that could be included among the rules of the systems to better allow them to promptly fix the damage, even going as far as requesting help and having a clean-up stage before repair damage.

Of course with a better understanding of system-environment interaction we might end up realizing that it is not necessary to explicitly implement hard triggers and sequenced protocols, as these aspects of behavior will be implicitly evolvable. The following plots show the self-repair capabilities of the genome shown in Figure 4.4.2.

| 0 Steps | 192 Steps |
| 192 Steps (Damage Applied) | 384 Steps (another 192 Steps) |
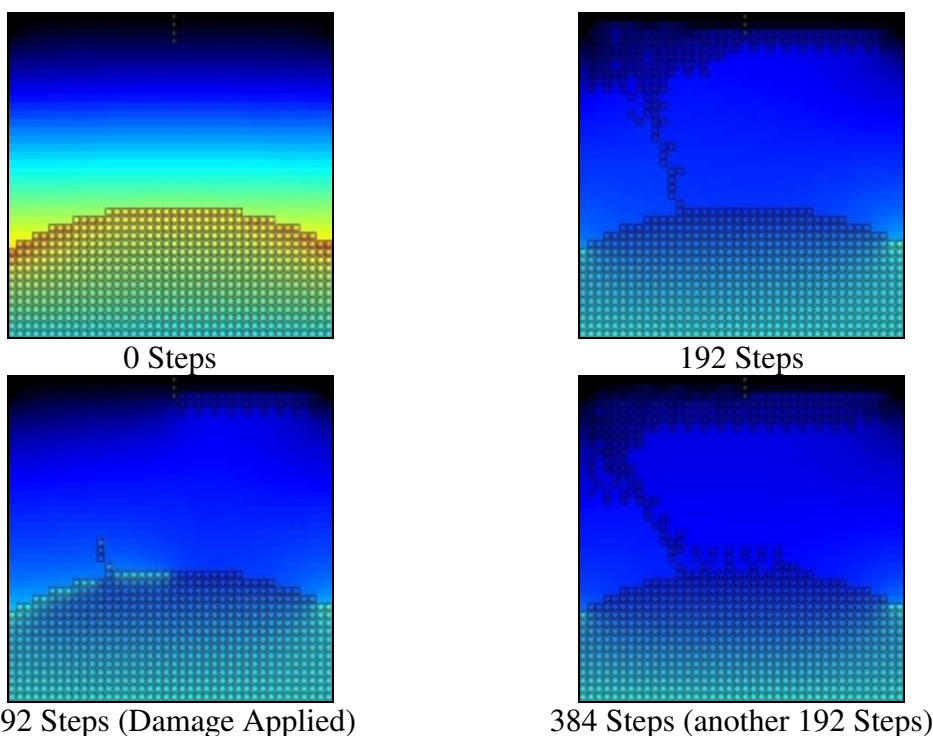
Figure 7.1.1 Self-Repair in Test 1: Agent rebuilds the part of the structure damage at step 192. It is not a exact rebuild but the principle is retained.

In Figure 7.1.1, it can be seen how the agent rebuilds the column and part of the canopy. It does not rebuild exactly the same thing but it still has the same idea. This is reminiscent of regeneration which is a property that some biological developmental systems have where a fully developed organism can replace lost parts (Wolpert et al. 236, 447-448).

| 0 Steps | 192 Steps |

| 192 Steps (Damage Applied) | 384 Steps (another 192 Steps) |
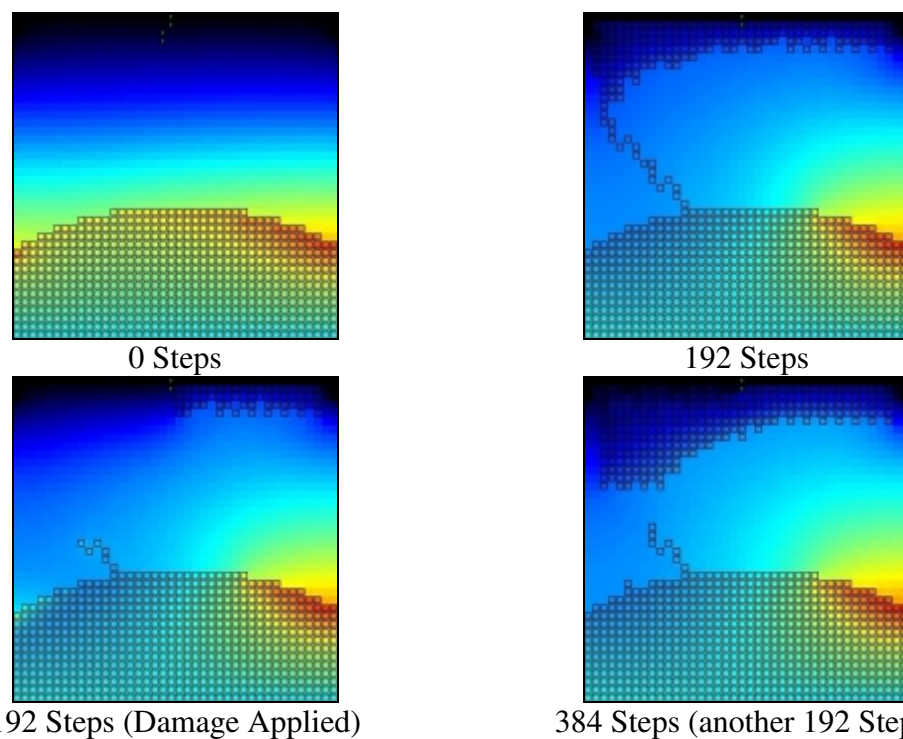
Figure 7.1.2 Self-Repair in Test 2: Agent is not able to completely repair the previous structure but it does repair the canopy and recovers the temperature profile (reduce loss of heat to sky using the canopy).

In Figure 7.1.2, the fact that the agent does not recover the vertical column that connects the canopy to the ground but it does recover the functionality by sort of building down in that direction and even partially rebuilding the canopy seems to

slightly undermine the need for a vertical column, yet it is still rebuilt eventually after working down from the canopy.

0 Steps	192 Steps
192 Steps (Damage Applied)	384 Steps (another 192 Steps)
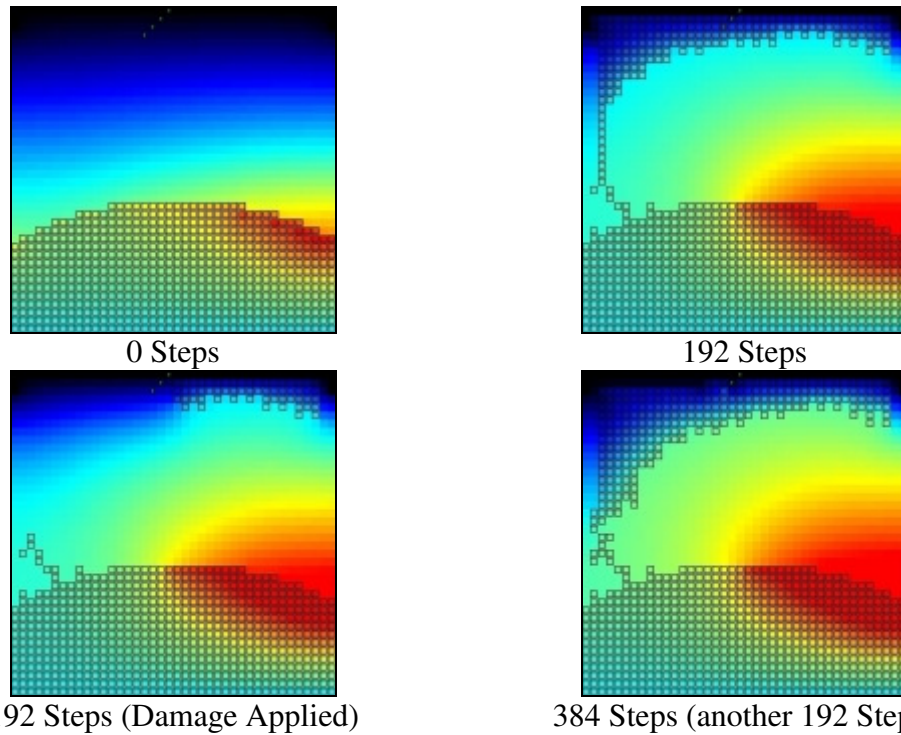
Figure 7.1.3 Self-Repair in Test 3: Canopy is worked down to the remains of the column.

This case seems to fall in between the first two where, as in the first test, the canopy is reconnected to the column and the canopy was worked down from the left-side down to the column, as in the second test. The first test was the only one where the vertical column was immediately worked up towards the canopy. And in tests 2 and 3 the canopy was built downwards from the left until at some point the vertical column shoot up to meet it. Although in the second test the column did not have enough time to connect.

CHAPTER 8

CONCLUSION

In this thesis it has been shown by experimental work that systems evolved using and interactive level of system-environment interaction where able to build more functional structures over different environmental conditions, showing a more robust behavior. Increased adaptability is very important when designing systems that will need to build structure in unforeseen environments or if one knows that the system is going need to perform it duties over a wide range of environmental conditions.

When working with developmental systems it is important to pay attention to the level of system-environment interaction present, especially to identify the role that the environment plays in the development of the system. In biology the important role of the environment in the development of an organism is yet to become widely recognized.

One of the most important issues in pre-modern biology was the struggle between the preformationist and epigenetic theories of development. The preformationist view was that the adult organism was contained in the sperm and that development was the growth and solidification of this miniature being. The theory of epigenesis was that the organism was not yet formed in the fertilized egg, but that it arose as a consequence of profound changes in shape and form during the course of embryogenesis. It is usually said that the epigenetic view decisively defeated preformationism. Yet it really preformationism that has triumphed, for there is no essential difference between the view that the organism is already formed in the fertilized egg and

the view that the complete blueprint of the organism and all information necessary to specific it is contained there, a view that dominates modern studies of development. (Lewontin, 5-6)

The importance of looking closely at the system-environment interaction in system that we work with goes beyond just achieving better performance and adaptability, it is also about having a better understanding of the behavior of our experimental system and seeing the importance of the role that the environment plays in system development.

8.1   Nature and Stigmergy

This thesis work was introduced by talking about the role of nature in it inspiration. However the work itself is about developmental theory and the terms systems, environment, and development take on a more abstract shape as references to nature and termites quickly become sparse. The discussion was kept away from termites and nature to avoid confusion about its claims being made in relation to termites or social insects. This is due to the focus of the work being more on engineering and computer science related. However, it must be clear that the author always kept the concept of termites close at hand in his mind when designing the experiments and writing the programming code. The agents where referred to as termites and the structures were referred to as mounds. The concept of stigmergy, a notion introduced to describe the cooperative behavior in termites. Is very key to this work and describes and interactive level of system-environment interaction among termites. The relation lies in that stigmergy is defined as:

The indirect communication taking place among individual termites through dynamically evolving features of a structure. (Bonabeau et al. 188-193)

The very notion of stigmergy is about cooperation between systems, a method of cooperation only possible at an interactive level system of system-environment interaction. This is because interactive systems are able to exploit changes in the dynamics of the environment; this ability in termites is what drives their stigmergic behavior and the inspiration for this work.

8.2   Future Work

The systems used in this work were designed as experiments to test a hypothesis. The high amount of simplification done on our test systems left us without any basis to make claims about the systems on which they where originally based on.

Seeing as how this thesis works on concepts taken from biology but applies them to engineering, it is logical to picture two direction in which this work could branch out. Biologically focused work stemming from this thesis may involve creating more complicated simulations to explore an learn more about termite behavior building mound in three-dimensions and embedding more physical properties into the environment such as basic fluid dynamics (Stam), mechanical properties to the materials, and pheromones for the surface. Adding pheromones to otherwise random simulations of ant movement has been shown to produce foraging patterns like those of several species of ants in Deneubourg's work. Perhaps giving the system more complicated goals besides temperature, such as relative humidity or making more complicated agents that behave differently as a factor of lighting, age/time, or hunger/power. Such factors can greatly contribute to the nest's overall structure as is

the case in Florida harvester ants (Tschinkel). A motivation to adding more realism to the models is to evolve simulated systems that we could ultimately port to real systems and use to build real-world structures. Using continuous systems in the simulation would be part of that step toward a physical realization of our evolved systems.

Further work expanding the quantitative metric presented as the Interactive Dynamics Index to include and classify all the different levels of system environment interaction mentioned in section 1.3 under one all-inclusive scale.

If even the simple setup used for these experiments is able to show self-repair and even when it wasn't even evolved for it, then more work exploring the self-repair capabilities of an interactive system must be explored. The ability to self-repair can be integrated in a fitness function and some tweak could be made in the environment to make self-repair especially worth while.

There is also potential in combining these self-repair capabilities with 3D printing technologies. Work is already being done in printing functional components with solid freeform fabrication using computer 3D models as blueprints (Malone). Such 3D-printers fitted with sensors and rules instead of conventional blueprints could be the first to build real-life structures using functional blueprints. With added mobility this technology could eventually become the platform for the robots described in the self-repair scenario given in section 7.1. Already real-life reactive developmental systems have been built (Werfel et al.) so interactive systems is the next logical step.

From an engineering point of view it would be of interest to design interactive developmental systems tasked with solving real-world engineering problems starting in areas where evolutionary algorithms are being used for design already.

There is great potential in the current work being done involving real-world systems designed for various tasks (Lipson and Pollack 974-978). In some cases the

systems are designed using evolutionary algorithms (Hornby and Pollack 223-246) and in other cases their behavior is (Lipson et al. 11-18).

These concepts are potentially very useful for applications such as cooperation in multiple-robot systems tasked with building adaptive structures in orbit, on other planets, or even the orbits of other planets.

//Main Code for single evaluation of reactive system

```
1    for (double m = 0; m <= 1; m += 0.5)
2    {
3        model.Initialize(m);
4        for (int i=0; i<lsim; i++)
5        {
6            model.ExecuteAgent(&individualgenome, agentsize);
7            //model.Run(runsteps, m);
8        }
9        model.Run(runsteps0, m);
10       RedrawViewsNow();
11       model.AgentFitness(targettemp);
12   }
```

Walkthrough:

(line 1)   Variable **m** used as input to indicate which test is being done. The 'for-loop' run three times for **m** equal 0, 0.5, and 1, which are actually the inverse values of the slope that the sunrays angles are going to be.

(line 3)   **model.Initialize(…)** – Initializes the model and uses **m** as input in order to set a specific temperature distribution.

(line 4)   **lsim** indicates length of simulation how many agents step are done. For every experiment discussed this was 128.

(line 6)   **model.ExecuteAgent(…)** – Executes rules of specified individual. Input are the genome with the rules and a varible that specifies number of rules.

(line 7)   Line used only for interactive system runs.

(line 9)   Line used only for reactive system runs. Executes environment update, agent never sees this. Only done for fitness evaluation.

(line 11)   **model.AgentFitness(…)** – Call to fitness function. Input is target temperature.

//Main Code for single evaluation of interactive system

```
1    for (double m = 0; m <= 1; m += 0.5)
2    {
3       model.Initialize(m);
4       for (int i=0; i<lsim; i++)
5       {
6          model.ExecuteAgent(&individualgenome, agentsize);
7          model.Run(runsteps, m);
8       }
9       //model.Run(runsteps0, m);
10      RedrawViewsNow();
11      model.AgentFitness(targettemp);
12   }
```

Walkthrough:

(line 1)   Variable **m** used as input to indicate which test is being done. The 'for-loop' run three times for **m** equal 0, 0.5, and 1, which are actually the inverse values of the slope that the sunrays angles are going to be.

(line 3)   **model.Initialize(…)** – Initializes the model and uses **m** as input in order to set a specific temperature distribution.

(line 4)   **lsim** indicates length of simulation how many agents step are done. For every experiment discussed this was 128.

(line 6)   **model.ExecuteAgent(…)** – Executes rules of specified individual. Input are the genome with the rules and a varible that specifies number of rules.

(line 7)   Line used only for interactive system runs. Executes environment update and is call after every rule execution. This allows the agent to use dynamic environmental feedback.

(line 9)   Line used only for reactive system runs.

(line 11) **model.AgentFitness(…)** – Call to fitness function. Input is target temperature.

//Code for ExecuteAgent(…) Part 1: Finding Action Site

```
for (int ix = 2; ix<nx-2; ix++)
{
   for (int iy = 2; iy<ny-2; iy++) //nested for-loops scan
environment
   {
      ax=ix; ay=iy;
      if (cell is empty)
      {
         if (cell has solid neighbors)
         {

//Collect sensor data
            double temp_avg = average temperature of 8 cells;
            double matt_avg = average density of 8 cells;

            double matt_gx=0;
            double matt_gy=0;
            double matt_angle;
            for (int i=-1; i<2; i++) {
               for  (int j=-1; j<2; j++) {
                  matt_gx += i*c(ax+i,ay+j).v;
                  matt_gy += j*c(ax+i,ay+j).v;
               }
            }
            matt_angle = atan2(matt_gy, matt_gx) * 180 / pi;
//gradient

//check sensor data distance to rules
            double dist=0;
            for (int i=0; i<agentsize; i++)
            {
               dist += sqrt(
               pow((pAgent->rule[i].temp_mean   - temp_avg),2) +
               pow((pAgent->rule[i].matter_mean - matt_avg),2) );
            }
            if ( dist < min_dist ) //save current data
            {
               min_dist = dist;
               rx=ax;
               ry=ay;
               tangle = matt_angle;
               for (int i=0; i<agentsize; i++)
               {
                  pAgent->rule[i].weight = 1/sqrt(
                     pow((pAgent->rule[i].temp_mean   -temp_avg),2)+
                     pow((pAgent->rule[i].matter_mean -matt_avg),2));
               }
            }
         }
      }
   }
}
ax=rx;   ay=ry; // coordinates for the action site saved
```

//Code for ExecuteAgent(…) Part 2: Combining Rules

```
//Once action site is selected the rules are combined
double rule_action = 0;
double rule_angle = 0;
double rule_angle_cos = 0;
double rule_angle_sin = 0;
double rule_dir = 0;
rn=0;

for (int i=0; i<agentsize; i++)
{
   rn += rule[i].weight; //rn = total weights of rules
}
for (int i=0; i<agentsize; i++)
{
   rule[i].weight /= rn; //using rn weight are normalized
//using weight all rules are added together
   rule[i].usecount += rule[i].weight;
   rule_action += rule[i].weight*rule[i].action;
//angles combined as vector components
   rule_angle_cos+= rule[i].weight*cos( rule[i].angle);
   rule_angle_sin+= rule[i].weight*sin( rule[i].angle);
   rule_dir    += rule[i].weight*rule[i].dir;
}
//vectors components then are turn into angle for new rule
rule_angle = atan2(rule_angle_sin, rule_angle_cos);

//once a rule has been formed from the other it is executed
if (rule_action >= 0.5)
{
   Deposit(rule_angle + tangle, signval(rule_dir));
}
else if (rule_action < 0.5)
{
   Remove (rule_angle + tangle, signval(rule_dir));
}
```

//Code for Deposit(…);

```
int n = 0;
for (int i=0; i<8; i++) //repeats 8 times going through all neighbors
{
   double angle = angle_in +(45*i*dir); //offset and rotation
   if (cell chosen is empty)
   {
     make cell solid;
     n = 1;
     break;
   }
}
//if all cell surrouding home are solid then no action has been taken
at this point, so then home cell is turned solid
if (n == 0)
{
   make home cell solid;
}
```

//Code for Remove(…);

```
for (int i=0; i<8; i++) //repeats 8 times going through all neighbors
{
   double angle = angle_in +(45*i*dir); //offset and rotation
   if (cell chosen is solid)
   {
     make cell empty;
     n = 1;
     break;
   }
}
//action will always been taken because a home cell always has at
least one solid cell neighbor
```

```
//Code for Run(…)    Part 1: Sunrays

double increment = 0.75; //sun heat intensity
double lres=0.25; //resolution of sunrays (only used when m is not 0)

if(m==0)
{
   for (int i=nx-1; i>=0; i--) {         //sideway scan
      for (int j=ny-1; j>=0; j--) {     //top to bottom scan
         if (top to bottom scan hits a solid cell) {
            cell_temp += increment;
            break;
         }
      }
   }
}

else if(m>0){
   m = 1/m;
   for(double b = -ny/m; b<nx; b+=lres){  //sideway scan
      for(int y=ny-1; y>0; y--){    //top to bottom scan
         int x = int (y/m + b);       //x = ray location
         if( x hits a solid cell){
            cell_temp += increment*lres;
            break;
         }
      }
   }
}

else if(m<0){
   m = 1/m;
   for(double b = 0; b<(nx-ny/m); b+=lres){  //sideway scan
      for(int y=ny; y>0; y--){       //top to bottom scan
         int x = int (y/m + b);       //x = ray location
         if(x hits a solid cell) {
            cell_temp += increment*lres;
            break;
         }
      }
   }
}
```

```
//Code for Run(…) Part 2: conduction physics update


double coeffs [2][2];
   coeffs[solid][solid]=0.025;
   coeffs[solid][empty]=0.05;
   coeffs[empty][solid]=0.05;
   coeffs[empty][empty]=0.25;

for (int i=0; i<nx; i++) {          // Temperature diffusion
   for (int j=0;j<ny; j++) {
      new_temp = old_temp +
      coeffs[home][neighbor1]*( neighbor1_Temp – home_Temp) +
      coeffs[home][neighbor2]*( neighbor2_Temp – home_Temp) +
      coeffs[home][neighbor3]*( neighbor3_Temp – home_Temp) +
      coeffs[home][neighbor4]*( neighbor4_Temp – home_Temp);
   }
}
//updates temperature by adding temperature differences scaled by
conduction coefficients which depends on both home and neighbors
matter states



//Code for AgentFitness(…)

int counter = 0;
double temp = 0;
//for-loops scan whole environment
for (int i=int(0.0*nx); i<int(1.0*nx); i++) {
   for (int j=int(0.0*ny);j<int(1.0*ny); j++) {
        counter++;
        //temp will be total temperature difference
        temp += fabs(targett – cell_temp);
      }
   }
}

fitness += 2*(50 – temp/gas);//3;
             }
```

REFERENCES

Almássy, Nikolaus P.W., Erik Vinkhuyzen. "Evolution of Adaptive Behavior in Dynamic Environments." In Intelligent Automation and Soft Computing: Trends in Research, Development and Applications. Ed. M. Jamashidi, C.C. Nguyen, R. Lumia, and J. Yuh. Albuquerque, NM: TSI Press, 1994.

Bard, Jonathan B. L.. Morphogenesis: The Cellular and Molecular Processes of Developmental Anatomy. UK: Cambridge University Press, 1992.

Bentley, Katie, and Chris Clack. "Morphological Plasticity: Environmentally Driven Morphogenesis." ECAL (2005): 118-127.

Bentley, Peter J.. "Fractal Proteins." Genetic Programming and Evolvable Machines 5.1(2004): 71-101.

Bentley, Peter J., and Sanjeev Kumar. "Three Ways to Grow- Designs: A Comparison of Evolved Embryogenies for a Design Problem." GECCO 1(1999): 35-43.

Bonabeau, Eric, et al. "Self-Organization in Social Insects." TREE 12.5(1997): 188-193.

Bonabeau, Eric, Sylvain Guerin, Dominique Snyers, Pascale Kuntz, and Guy Theraulaz. "Three-dimensional architectures grown by simple `stigmergic' agents." BioSystems. 56(2000): 13-32.

Bongard, Joshua, and Hod Lipson. "Automated robot function recovery after unanticipated failure or environmental change using a minimum of hardware trials." Proceedings NASA/DoD Conference on Evolvable Hardware (2004): 169- 176.

Bongard, Josh C., and Rolf Pfeifer. "Evolving Complete Agents Using Artificial Ontogeny." In Morpho-functional Machines: The New Species (Designing Embodied Intelligence) Berlin: Springer-Verlag. (2003): 237-258.

Bongard, Joshua, and Hod Lipson. "Integrated Design, Deployment and Inference for Robot Ecologies." Proceedings of Robosphere (2004)

Bongard, Josh C., and Rolf Pfeifer. "Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny." GECCO (2001): 829-836.

de Garis, Hugo. "Artificial Embryology: The Genetic Programming of an Artificial Embryo." In Artificial Life III Workshop, Santa Fe, New Mexico, USA, June 1992.

Deneubourg, J. L., and S. Goss. "Collective Patterns and Decision Making." <u>Ethology, Ecology, and Evolution</u> 1(1989): 295-311.

Deneubourg, J. L., S. Goss , N. Franks, and J. M. Pasteels. "The Blind Leading the Blind: Modeling Chemically Mediated Army Ant Raid Patterns." <u>Journal of Insect Behavior.</u> 2.5(1989): 719-725.

Eggenberger, Peter. "Evolving Morphologies of Simulated 3d Organisms Based on Differential Gene Expression." 4th European Conference on Artificial Life (1997): 205-213.

Emerson, Alfred E.. "Termite Nests–A Study of the Phylogeny of Behavior." <u>Ecological Monographs</u> 8(1936): 247-284.

Francu, Andreea. "Plant Modeling." Andreea Francu's Home Page. 19 May 2006. <http://andreea.francu.com/lgrammar/fractals.html>.

Harris, W. Victor. <u>Termites; Their Recognition and Control</u>. London: Longmans, 1961.

Hemberg, Martin, Una-May O'Reilly. "Extending Grammatical Evolution to Evolve Digital Surfaces with Genr8." <u>EuroGP</u> (2004): 299-308.

Hornby, Gregory S.. "Functional Scalability through Generative Representations: the Evolution of Table Designs." <u>Environment and Planning B</u> 31.4(2004): 569-587.

Hornby, Gregory S., Hod Lipson, and Jordan B. Pollack. "Evolution of Generative Design Systems for Modular Physical Robots." <u>Proceedings IEEE International Conference on Robotics and Automation</u> 4(2001): 4146- 4151.

Hornby, Gregory S., and Jordan B. Pollack. "Body-Brain Co-evolution Using L-systems as a Generative Encoding." <u>GECCO</u> (2001): 868-875.

Hornby, Gregory S., and Jordan B. Pollack. "Creating High-Level Components with a Generative Representation for Body-Brain Evolution." Artificial Life 8.3(2002): 223-246.

Hornby, Gregory S., and Jordan B. Pollack. "Evolving Complete Agents Using Artificial Ontogeny." <u>Computers & Graphics</u> 25(2001): 1041–1048.

Incropera, Frank, and David DeWitt. <u>Fundamentals of Heat and Mass Transfer</u>. 4th ed. New York: John Wiley & Sons, 1996.

Koza, John R.. <u>Genetic Programming: On the Programming of Computers by Natural Selection</u>. Cambridge, MA: MIT Press, 1992.

Kumar, Sanjeev. "Investigating Models of Development for the Construction of Shape and Form." <u>Doctoral Thesis, UCL</u> (2004).

Kumar, Sanjeev, and Peter J. Bentley. <u>Computational embryology: past, present and future</u>. In: Advances in evolutionary computing: theory and applications. New York: Springer-Verlag, 2003.

Kumar, Sanjeev, and Peter J. Bentley. <u>On Growth, Form and Computers</u>. London: Academic Press, 2003.

Lewontin, Richard. <u>The Triple Helix</u>. Cambridge, Massachusetts: Harvard University Press, 2000.

Lindenmayer, A., and G. Rozenburg. <u>Automata, Language, Development</u>. Amsterdam: The Netherlands: North-Holland, 1976.

Lipson, Hod. "Evolutionary Robotics and Open-Ended Design Automation." <u>Biomimetics: Biologically Inspired Technologies</u>. Ed. Yoseph Bar-Cohen. Boca Raton, FL: CRC Press, 2006.

Lipson, Hod, Joshua Bongard, Victor Zykov, and Evan Malone. "Evolutionary Robotics for Legged Machines: From Simulation to Physical Reality." <u>Proceedings of the 9th Int. Conference on Intelligent Autonomous Systems.</u> (2006): 11-18.

Lipson, Hod, and Jordan B. Pollack. "Automatic Design and Manufacture of Artificial Lifeforms." <u>Nature</u> 406(2000): 974-978.

Mahdavi, Siavash H., and Peter J. Bentley. "An Evolutionary Approach to Damage Recovery of Robot Motion with Muscles." <u>In 7th European Conference on Artificial Life</u> (2003): 248-255.

Malone, Evan. "Functional Freeform Fabrication for Physical Artificial Life." <u>Proceedings ALIFE 9</u> (2004): 100-105.

Mason, Zachary. "Programming with Stigmergy: Using Swarms for Construction." <u>ALIFE8</u> (2002): 371-374.

Miller, Julian F.. "Evolving a self-repairing, self-regulating French flag organism." <u>GECCO</u> (2004).

Miller, Julian F.. "Evolving Developmental Programs for Adaptation, Morphogenesis, and Self-Repair." Proceedings of the 7th European Conf. on Advances in Artificial Life 2801(2003): 256-265.

Mitchell, Melanie. An Introduction to Genetic Algorithms. MIT Press, 1998.

Murata, S., H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. "A 3-D self-reconfigurable structure." ICRA (1998): 432-439.

Nagpal, Radhika. "Programmable Self-Assembly using Biologically-Inspired Multiagent Control." AAMAS (2002).

Nagpal, Radhika, Attila Kondacs, and Catherine Chang. "Programming Methodology for Biologically-Inspired Self-Assembling Systems." AAAI Spring (2003).

Pfeifer, Rolf, and Fumiya Iida. "Morphological computation: Connecting body,brain and environment." Japanese Scientific Monthly 58.2(2005): 48–54.

Prusinkiewicz, Przemyslaw, and Aristid Lindenmayer. The Algorithmic Beauty of Plants. New York: Springer-Verlag, 1996.

Quick, Tom, Chrystopher L. Nehaniv, Kerstin Dautenhahn, and Graham Roberts. "Evolving embodied genetic regulatory network-driven control systems." ECAL. (2003):

Rieffel, John. "Evolutionary Fabrication: The Co-Evolution of Form and Formation." Doctoral Thesis, Brandeis University (2006).

Rieffel, John, and Jordan B. Pollack."Automated Assembly as Situated Development: Using Artificial Ontogenies to Evolve Buildable 3D Objects." GECCO (2005): 99-106.

Rudge, Timothy, and Nic Geard. "Evolving Gene Regulatory Networks for Cellular Morphogenesis." Recent Advances in Artificial Life. Ed. H. Abbass, T. Bossamaier, and J. Wiles. Singapore: World Scientific Publishing, 2005.

Stam, Jos. "Real-Time Fluid Dynamics for Games." Proceedings of the Game Developer Conference (2003).

Stanley, Kenneth O., and Risto Miikkulainen. "A Taxonomy for Artificial Embryogeny." Artificial Life 9.2(2003): 93-130.

Tschinkel, Walter R.. "The nest architecture of the Florida harvester ant, *Pogonomyrmex badius*." Journal of Insect Science 4:21(2004): 1-19.

Werfel, Justin, Yaneer Bar-Yamyz, Daniela Rus, and Radhika Nagpalz. "Distributed Construction by Mobile Robots with Enhanced Building Blocks." <u>Proceedings IEEE International Conference on Robotics and Automation.</u> (2006): 2787-2794.

Wolpert, Lewis, Rosa Beddington, Thomas Jessell, Peter Lawrence, Elliot Meyerowitz, and Jim Smith. <u>Principles of Development</u>. 2nd. New York: Oxford University Press, 2002.