1999

Artificial Societies: A Computational Model of Disease Transmission

Nathan T. Moore
College of William & Mary - Arts & Sciences

Artificial Societies: A Computational Model of Disease Transmission

A Dissertation

Presented to

The Faculty of the Department of Computer Science

The College of William & Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Masters of Science

by

Nathan T. Moore

1999

APPROVAL SHEET

This dissertation is submitted in partial fulfillment of

the requirements for the degree of

Masters of Science

[illegible]

Nathan T. Moore

Approved, August 1999

[illegible]

Stephen Park
Thesis Advisor

[illegible]

Gianfranco Ciardo
Department of Computer Science

[illegible]

Rex Kincaid
Department of Mathematics

To my wife, who sweated it out with me, and Dr. Park for putting up with the many midnight drop-offs ...

Table of Contents

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Park for his time an patience in getting this dissertation finished.

I also want to thank my thesis committee for managing to bear with me through the scheduling of my oral defense, and Dr. Kincaid especially for reading this thesis over his vacation.

And, of course, my wife, who stood by encouraging me throughout the whole process.

List of Figures

ABSTRACT

Artificial societies are a growing field of research in several disciplines, including computer science. Artificial society models are used to investigate hypothesis which cannot be studied in a lab. In an artificial society, a population of agents act and interact consistent with a set of micro-scale rules. Researchers then look for macro-scale behavior resulting from the rules. Although most literature available on artificial societies does not focus on the artificial society model at the computational level, that is the focus of this dissertation. In this dissertation, a typical artificial society is created consistent with a simple set of rules to guide agent movement and reproduction. An additional set of rules is added which introduce immune systems, diseases and disease transmission. With these rules in place, sensitivity analysis is used to see how selected system parameters affect the stable agent population.

Artificial Societies: A Computational Model of Disease Transmission

Chapter 1

Introduction

The human immune system features a complex mix of genetics and biological learning. The immune system has a method for fending off invading microbes, both viruses and infectious bacteria. When exposed to invading diseases, the immune system will attempt to attack and suppress the disease. Often, the extremes — young and old — are more susceptible to disease because their immune system is underdeveloped or lacks the resources to cope. While not every disease or virus can be defeated by the immune system, the diseases a human survives are coded into the immune system in the form of specific anti-bodies. In this way, if the human encounters the disease again, the specific anti-bodies attack the disease and notify the immune system to increase anti-body production to eradicate the disease.

Disease transmission is just as complex as the immune system. Some diseases require a very specific path for infection, such as parasites or worms, where any break in the path will prevent an infection. There are other diseases like the common cold, which are able to spread easily and quickly without a specific infection path. A disease can often spread rapidly when there is a large population of hosts (those who can carry the disease) in close quarters. This common close proximity exposure provides ripe conditions for an epidemic to sweep through the host population. Children, because of their many playmates, their habit of putting objects into their mouth, and their underdeveloped immune systems, are vectors for diseases. Children easily catch a disease, and then spread it to their playmates and family.

As a society clusters into groups, the individual immune systems are able to act together for the benefit of the group. For example, if a certain fraction of the population is immune to a disease, the disease may not be able to survive in the population. Thus, the entire population benefits from an immunity that only some of the population possess [2].

Modeling an immune system and disease transmission presents a challenging task. Diseases are capable of producing a multitude of effects which run the gamut of severity. The tasks of the immune system are to "remember" (encode) diseases, to adapt when exposed to new diseases, and, when infected again, "recall" the disease and attack it. The process of encoding and remembering diseases is complex and is not readily simplified. Several attempts have been made to use differential equations to model an immune system, with varying degrees of success. The most notable attempt is the S-I-R model [2]. This model involves "susceptibles," "infectives," and "removed" populations. Susceptibles are disease free, infectives are active carriers, and removed are "quarantined" infectives. Individuals are added to the removed population at some rate proportional to their numbers. The flow of individuals from population to population is then modeled using differential equations [2]. The SIR model does not deal with an immune system explicitly.

The use of agent-based computational models in social sciences is not new. The idea first came about in the early 1970's, and has become more mainstream in recent years [1][3]. An early social science study started with a "checker-board" model developed by Thomas Schelling. In Schelling's experiments, blue and red checkers moved about a grid in an attempt to model self-segregation [1][2]. Since then, many mathematical models have been conceived to try and define social processes. Often, the mathematical complexity of the equations needed to solve the models was so great that simulation became the natural alternative [3].

The study of agent-based models offers the social sciences an environment in which a hypothesis can be tested. For example, one cannot grow a population of humans in a lab. However, one can attempt to model human interaction on several levels by using agents, and try to determine any significant macroscopic behavior produced by a population of agents guided by microscale rules. Agent-based models of social processes are often termed "Artificial Societies"[2][1]. The agents exist in an environment typically represented by a cellular automata, and referred to as the landscape. The use of a cellular automata landscape in an artificial society model has a basis in the type of agent interactions, and the discretisation of time. The cellular automata landscape also helps to identify emerging macro behavior [4].

The use of an artificial society model to simulate human behavior has its share of skeptics. The major criticism is that humans are too complex to be simulated and that the results can lead to interpretations that are questionable at best [1]. However, the major advantage of using an artificial society model is that, "one can ... simulate the social process of interest in the computer and ... carry out experiments that would otherwise be quite

[1]Axtell gives credit to Builder and Banks [1991] for coining the term.

impossible" [3].

An important question in artificial society simulation studies is the validation of the model [3]. One can devise a model and produce simulation-based results, but how does one know the results are valid? The best "validation" may come from the basis of the model — people themselves. Gilbert reports having built several artificial society models where the model was based upon data gathered from personal interviews. The results of the simulation were then compared with the interviews [3].

This paper presents an agent-based computational artificial society model where agents move about a landscape, consume resources, procreate, and communicate diseases to one another. This model differs significantly from an ordinary differential equations approach. The ODE approach focuses on agent populations and the flow of agents from one population type to another. In contrast, the agent-based model deals with immune systems and diseases explicitly. The learning process of the agent's immune system is investigated, along with the communication of diseases.

One or more resources are the main characteristic of the landscape, which contains and often grows the resources. These resources may be defined as anything which increases an agents chance at reproduction [6]: for example, food, or in more detailed models, money. A unifying trait of a resource is that it can be consumed by an agent, and once consumed the resource is no longer usable by other agents [6]. Because an agent will die if it does not gather sufficient resources, a motivating factor for the agents is the need to travel about the landscape and collect resources. In the model presented here, this resource acts as both a food source and a metric for reproduction.

Evolution, or "learning" by agents in an artificial society is not a new concept. Gilbert describes his attempts to use "genetic algorithms" in his simulations to allow his agents to learn and evolve [3]. Genetic algorithms often use a "fitness" function to determine if a change is an improvement or not. In an artificial society model, an immune system is considered more fit if it has learned a disease and is therefore no longer susceptible to the disease. This concept of learning, or evolving, is an important part of the artificial society model used in this dissertation.

Agent reproduction goes hand in hand with evolution in that agents will pass on traits to their offspring. Macy presents an artificial society model where agents "learn" through evolution and procreation [5]. Agents with traits that make them more likely to survive should be more likely to reproduce than agents who have poorer traits or fewer resources. In time, this should produce a more fit agent population.

This dissertation is not a social science study. Instead, the focus is on the development of the simulation model and a systemic study of how the model is sensitive to changes in key model parameters. I take a scientific look at the model and its underlying assumptions and attempt to quantify how those assumptions affect the behavior of the model. I make no attempt to characterize any social ordering or other social phenomena which may be produced by the model. The artificial society model I used is based upon my interpretation of the model presented in *Growing Artificial Societies* [2], with several modifications.

This dissertation is organized in such a way as to carefully develop and document the artificial society model. Chapter two exclusively deals with the model at three levels — conceptual, specification and implementation. Chapter three discusses the computational experiments, and the underlying reasoning behind why I chose these experiments. Computational experiments lead to the collection of data, and chapter four is where I present the data I gathered, as well as my analysis of the data. I conclude with my ideas on the model, conclusions about the model based on the experiments, as well as ideas for further research.

Chapter 2

Model

As advocated in [7], model development should take place at three levels — the conceptual, the specification and the algorithmic level. At the conceptual level, the ideas of the model are presented, and a general framework of the model is explained. With a conceptual model in place, the model specifications are developed. The specifications build on the generality of the model at the conceptual level by adding specific details. By building on the specification model, the computational model provides the actual implementation details, including data structures, algorithms, and other implementation specific information [7].

2.1 Conceptual Level

The model presented in this paper is an artificial society simulation model where a population of agents live on a landscape and, as time evolves, interact with both the landscape and other agents. This is a dynamic model with stochastic elements, where time evolves discretely. At the conceptual level, I introduce the agents and the landscape, as well as the rules which guide them, as follows.

2.1.1 Rules

Rules guide the actions and interactions of all the agents. There are three types of actions and interactions. Agents interact with the landscape when they gather the resource at a landscape location and when an agent is born. The landscape interacts with itself when the landscape re-grows resource. Agents interact with other agents when they reproduce and when they spread disease. Micro-scale rules govern all these actions and interactions. Traditional wisdom in artificial society models is that these rules should be simple to explain, and simple to implement. Because all activity which takes place on the landscape follows the rules, the rules drive all agent behavior.

2.1.2 Landscape

The landscape is the source of resource for the agents. The landscape is modeled as a cellular automaton. A landscape location is characterized by its level of resource, and whether an agent currently occupies the location. Both the occupancy and the resource level change as time evolves. As time evolves, each landscape location "grows" resource at a constant rate, up to a maximum level of resource (the "capacity" of the location), or until an agent harvests the resource at the location. Each time an agent occupies a landscape location, it harvests all of the resource currently there.

2.1.3 Agents

As time evolves, agents move around the landscape harvesting and consuming resource. Agents have a set of states which evolve with time, and a set of static attributes. Agents have a level of resource they have gathered, and they consume their stores of resource at a constant rate (their metabolism). As time evolves, agents age, move, interact with one another, harvest resource and consume resource. When agents move, they only consider a finite number of landscape locations proximate to their current location. Agents may carry diseases which they transmit to other agents. Agents age as time progresses, and, unless they die of starvation, will die a natural death after a certain amount of time.

2.1.4 Immune System

Each agent has an immune system, which evolves with time. The immune system's job is to recognize a disease, attack the disease, and remember the disease [2]. When an agent becomes infected, the immune system "fights" the disease, which raises the agents metabolism and causes the agent to consume resource at a higher rate. This increase in metabolism (the "burden") is intuitive in that a disease takes away resources from a host to replicate when the disease infects a host. The immune system attempts to encode or "learn" a disease such that the agent will remember the disease in the future. When an agent becomes infected, there are only two outcomes — either the agent will die because of the increased burden, or the agent's immune system will learn and defeat the disease.

2.1.5 Diseases

Each agent is created with a set of diseases. These diseases do nothing to the agent other than increase the agent's metabolism (while the immune system "fights" the disease). Agents communicate diseases with other agents as time evolves. Unlike the immune system, diseases do not change as time evolves. When agents are in close proximity with one another, they will infect one another. When an agent contracts a disease, a certain amount of time will be needed before the agent is well again, and rid of the disease. This time is based upon the immune system, and how long it takes the immune system to learn the disease. When an agent has rid itself of a disease, its immune system will have changed to learn the disease.

2.1.6 Procreation

Agents interact with one another to produce new agents. Agents need to be fertile in order to reproduce. To be fertile, an agent needs to be of child bearing age, and have enough resource to support procreation. During procreation, the parent agents pass on their attributes to the children agents. Because the parent agents must be fertile, the child agent will have sufficient resources to not die upon birth. The inheritance of attributes from parents should lead to more "fit" children.

2.2 Specification Level

At the specification level, the agents and the landscape are both described in terms of their states and attributes. Time is introduced, and its role is discussed. Having only received a cursory introduction in the previous section, the diseases and the agent immune system are more fully explained in this section.

2.2.1 Time

Evolution on the landscape involves multiple actions and interactions taking place, in parallel, as time evolves in discrete time steps. Time is denoted as $t = 0, 1, 2 \ldots T$ where T is the stopping time. The actions and interactions involve agents moving, harvesting resource from the landscape, interacting with other agents and the landscape, the agent's immune system changing, and the landscape re-growing resource.

2.2.2 Landscape

As illustrated in figure 2.1, the landscape is a 2-dimensional $X \times Y$ grid of cells with periodic boundary conditions. What this means is that the North side connects with the South side, and the East side connects to the West side. In effect, this creates a torus shaped world with no "edges." Each cell is specified by an (x, y) coordinate. Each landscape cell can be characterized by a set of dynamic states, and static attributes. The states are the current resource level and the occupancy status. The attributes are the resource grow-back rate and the resource capacity. When an agent moves to a cell, it gathers all the resource at that cell. Accordingly, at that time, the cell's resource level is reduced to zero. Each cell re-grows resource at some rate $\alpha(x, y) > 0$. Each cell has a current continuous (real-valued) resource level $r(x, y)$ and a maximum capacity $c(x, y)$. At each time step, each cell will grow $\alpha(x, y)$ units of resource, up to its capacity. That is, at time t the resource level at cell (x, y) will be $\min\{r(x, y) + \alpha(x, y), c(x, y)\}$ where $r(x, y)$ is the resource level at time

$t - 1$. At any time, each landscape cell is either occupied or not; that is, multiple agents cannot occupy the same cell.

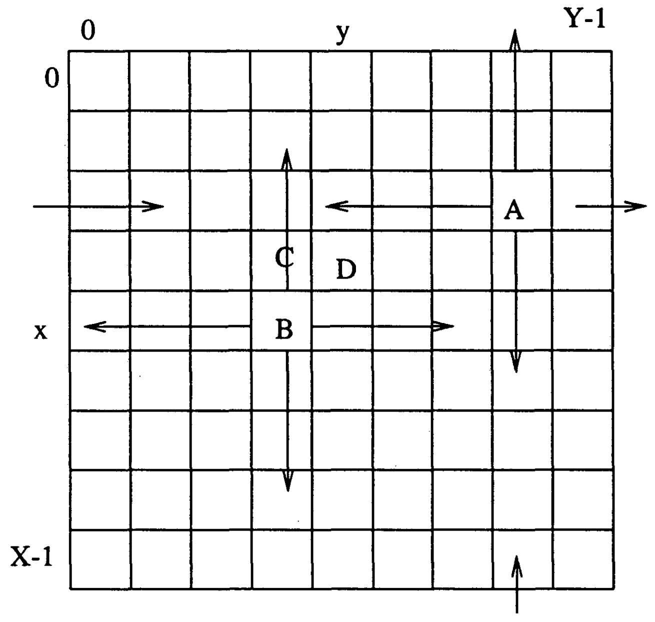

Figure 2.1: The landscape, FOV, and neighbors

2.2.3 Agents

As with the landscape, each agent has a set of dynamic states and static attributes. The agent attributes are a metabolism, a field of vision (FOV), a sex, a maturity age (when an agent becomes fertile), an infertility age, a maximum age (when an agent will die of old age), an initial wealth, and the maximum number of diseases the agent can carry. The agents states are its current wealth (accumulated landscape resource), age, immune system, the number of diseases with which the agent is infected, the diseases themselves, and the agent's (x, y) location on the landscape. The FOV is the set of landscape cells that the agent can see from its current location. An agent's sex is either male or female. The wealth and metabolism of the agent are real valued. Because time evolves discretely, the agent's age is integer valued. At $t = 0$, there is an initial population of A agents on the landscape.

An agent's field of view is limited to the north/south/east/west directions. Agents cannot see "off axis". This is shown in figure 2.1, where agent B is a can "see" agent C, but not agent D. (The FOV of agent B is demonstrated when agent B's FOV parameter is three.) In figure 2.1, the FOV for agent A demonstrates the wrap around effect.

A neighboring agent is one (of four possible) that is directly adjacent, either to the North, South, East or West. In figure 2.1, agent B and C are neighbors, and C and D are neighbors, but B and D are not neighbors.

Each time step, each agent attempts to move to the unoccupied landscape cell within its FOV with the highest resource level. If two or more cells have the same highest resource level, the agent will move to the closest cell. If all the cells within an agent's FOV are void of resource or occupied, the agent will not move. After an agent has moved, the agent will try to reproduce with each neighboring agent (if any). Following that, the agent will then give a disease to each neighbor (if any).

Although procreation is a desired behavior, disease transmission is an undesirable side effect of having agents in close quarters. Multiple reproductions per time step are possible in this model, though the cost of reproduction should limit such events.

If an agent's wealth ever drops to zero, the agent will die. Thus all agents are compelled to gather resource. An agent will also die if it reaches its maximum age ζ.

2.2.4 Immune System

To model the immune system for each agent, a simple "genetic" sequence is represented by a binary string of length I. Each disease is represented by a binary string of length D. A binary string is a string where each position is a "bit", either a one (on) or a zero (off). The immune system string-length I is considerably larger than the disease string-length D to allow for multiple diseases to be learned without significant overlap. Overlap occurs when two or more diseases an agent is immune to share multiple bits in the immune system. For example, consider the two diseases "00110" and "11011". The two diseases have the same three bits, "011", in the leading/trailing position. When the diseases are learned, the immune system may have a substring which looks like "0011011" which would grant immunity to both diseases. In this case, the overlap is good in that two diseases share some genetic components, and in that way overlap frees up other sections of the immune system to learn other diseases.

A disease is assumed to have only one effect on an agent, and that is to increase the infected agent's metabolism, thereby causing the agent to consume its wealth more rapidly. This metabolism increase is called the disease "burden". An agent is immune to a disease if and only if there is an exact match of the disease string to some substring in the agents immune system. If an infected agent is not immune to a disease, the agent's immune system will find the closest matching substring to the disease — the Hamming Distance [1]. The Hamming distance of the closest matching substring defines how long it will take the agent to become immune to the disease. That is, for each bit that is different from the disease

[1]The Hamming distance between two binary strings in obtained by comparing the strings position-by-position and totaling the number of positions where the strings differ. Two strings are identical if and only if they have a Hamming distance of zero [2].

string in the immune system's closest matching substring, one time step is needed to "learn" that bit of the disease. In this way, an agent will be "cured" of the disease, at which time the immune system will change the substring to be an exact match to the disease string. It is possible, when encoding for one disease, that immunity to a previously learned disease will be lost.

For example, consider a disease string "10100"($D = 5$) and an immune system string "1011010101" ($I = 10$). For each possible substring, the associated Hamming distance is shown:

start position	1	0	1	1	0	1	0	1	0	1	Hamming Distance
1	1	0	1	0	0						1
2		1	0	1	0	0					3
3			1	0	1	0	0				3
4				1	0	1	0	0			1
5					1	0	1	0	0		4
6						1	0	1	0	0	1

In this example, there are three closest matching substrings, each with a Hamming distance of one. In this model, the first closest matching substring "10110" is the immune system substring that will be changed to "10100" if the agent lives long enough to learn the disease.

Each agent (created or born) has an initial number of diseases. The initial diseases for each agent are created at random from the set of all possible diseases. Thus, at time $t = 0$, all of the A agents are given the same number of diseases, K. Because the set of all possible diseases is the set of all binary strings of length D, the size of this set is 2^D.

If an agent is carrying two diseases which have some shared overlap in the immune system, a race condition can occur. The first disease to be learned can be overwritten by the second disease to be learned. If this happens, the agent would then be susceptible to the first disease.

2.2.5 Disease Transmission

All diseases are highly infective. If an agent has any diseases, it will pass one randomly selected disease to each neighbor. Only one disease is shared between an agent and its neighbor per interaction. Consider figure 2.1 again. Agent B will give agent C one disease. Agent C will give agent B one disease, and agent D one disease. Since each agent has a maximum number of diseases it can carry (M), a transmission attempt may fail. Similarly, an agent cannot be infected more than once with the same disease at the same time and thus a transmission attempt will fail if the agent is give a disease it is currently carrying. If the agent is immune to the disease it is given, the transmission attempt will "succeed" but the "infected" agent will not carry the disease. A neighbor may try to give a disease it recently received back to the agent who gave it, either during the same time step or at a later time step.

An interesting effect here is that the shape of the landscape, and distribution of resource on the landscape in a sense drives the disease transmission model. Because agents will cluster in areas of high resource, agents in the high resource areas will have neighbors. Since the disease transmission mechanism relies heavily on neighbors, a high concentration of resource results in agents in close proximity sharing multiple diseases.

2.2.6 Procreation

When two fertile agents of opposite sex are neighbors, and there is an open cell adjoining one of the parents, a new agent will be born during that time step. An agent is considered fertile if it has reached maturity (*i.e.*, is old enough to reproduce), but not infertility (*i.e.*, is not too old to reproduce), and has at least as much wealth as its endowment. The wealth constraint deals with the parent being able to "pass on" resource to the child, and the child being able to live for at least a brief period of time even if there are no landscape resources available for the child to gather.

When a child is born, the child will inherit its attributes from its parents according to Mendel's laws of inheritance. That is, for each attribute, the child agent has an equal chance of inheriting the attribute from the mother or the father. All of the child's attributes are inherited from its parents — no "new" attributes are created. That is, there are no "crossovers" or mutations[2]. The child's initial wealth ("endowment") will be the same as the parent's. That is, all agents have the same endowment. The child's immune system is inherited from the parent's immune system. For each bit in the child's immune system, if both parents have the same value, the child inherits that value; otherwise the inherited bit is equally likely to be a 0 or a 1.[3] All children begins with the same number of diseases, K. By inheriting its immune system from its parents, the child's immune system hopefully learns the more "fit" bit-sequences/partial strings to have set, since the parents have lived long enough to have conquered some diseases, and perhaps certain partial strings are more

[2]Crossover is where a sequence, or part of a gene breaks apart and reforms, creating a new gene. Crossover has no meaning in this model.

[3]It should be noted that a learned disease will probably not occupy the same positions in each parent's immune system. Therefor, even if both parents are immune to a disease, their child may not be. This inheritance idea comes straight from Axtell & Epstein.

effective for warding off disease.

2.3 Computational Level

In this section I present the data structures used, and some pseudo code to explain the model at the computational level. All the abstractions from the previous sections are set into more concrete terms.

2.3.1 Third Party Software Used

The random number generator and associated library I used was developed by Steve Park [7]. This generator provides for multiple streams of random numbers, as well as many distributions.[4] In particular, I used the Uniform distribution and the Equilikely distribution. Both distributions take a range, and return a random variate within that range. The *Uniform(a,b)* function returns a real value between a and b, and the *Equilikely(a,b)* function returns an integer value between a and b. The Bernoulli random variable was also used, where the *Bernoulli(p)* function returns 1 with probability p and 0 with probability $1 - p$.

[4]Appendix A has more information about the random number generator

2.3.2 Model Implementation

At the computational level, the model is structured as follows: [5]

```
InitializeLandscape()
InitializeAgents()
For t = 0 to T
  MoveAllAgents()
  For each agent
    HarvestResource()
    Procreate()
    TransmitDiseases()
    UpdateAgents()
  End For
  RegrowLand()
  ShuffleAgents()
End For
```

In `InitializeLandscape()`, the landscape is initialized to have each cell unoccupied, have the growback rate $\alpha(x,y)$ be constant for all cells at 1.0, and have the cell capacity $c(x,y)$ set as follows.

$$c(x,y) = 4\ (C_1(x,y) + C_2(x,y))$$

$$C_1(x,y) = \exp\left(-\left(\frac{x - X/4}{\sigma_x}\right)^2 - \left(\frac{y - Y/4}{\sigma_y}\right)^2\right)$$

$$C_2(x,y) = \exp\left(-\left(\frac{x - 3X/4}{\sigma_x}\right)^2 - \left(\frac{y - 3Y/4}{\sigma_y}\right)^2\right)$$

with $\sigma_x = 0.3\ X$ and $\sigma_y = 0.3\ Y$. Each cell is initialized at full resource capacity.

This Gaussian two-peak equation is used for the cell capacity because the values returned seem to be an accurate representation of the landscape resource values given in "Growing Artificial Societies." An important distinction to be made, however, is that in my model,

[5]The ordering of each events (movement, harvesting, procreation, etc) is important. Further discussion on this issue can be found in chapter four.

the resource levels are real-valued, whereas in "Growing Artificial Societies" the resource levels are integer-valued.

With an understanding of the landscape, its states and attributes, creating a data structure to encapsulate the information is straightforward. The landscape data structure is as follows.

```
typedef struct {
  /* states */
  float sugar;          /* Current resource level r(x,y)       */
  struct agent *sim     /* Pointer to agent occupying cell     */
                        /* Null if unoccupied                  */
  /* Attributes */
  float capacity;       /* Maximum resource level c(x,y)       */
  int growth;           /* Growback rate alpha(x,y)            */
} cell_type;
```

In `InitializeAgents()` the initial population of $A = 400$ agents is created and placed at random on a $X \times Y = 50 \times 50$ landscape. There is nothing special about the number 400 as an initial population size and, as seen in chapter four, other values could be used with similar results. Axtell and Epstein used an initial population of 400 for most of their experiments. As for other specific values used for the agents attributes, all come from the base model in "Growing Artificial Societies"[2]. Each agent is initialized as follows:

1. Each agent is given a FOV parameter which is Equilikely(1,6).

2. Each agent is given a metabolism which is Uniform(1,4).

3. Each agent is randomly placed at an empty cell on the landscape.

4. Each agent is given a sex which is Bernoulli(0.5).

5. The infertility age for males is Equilikely(50,60) and for females, Equilikely(40,50).

6. Each agent is given a maturity age which is Equilikely(12,15), regardless of sex.

7. Each agent has a natural death age ζ which is Equilikely(60,80).

8. Each agent is given an I=64 bit immune system. I is varied during some experiments. Each bit of each agents immune system is Bernoulli(0.5).

9. Each agent is given $K = 4$ disease strings and, for each disease string, the $D = 8$ disease bits are each Bernoulli(0.5). D and K are varied during some experiments.

10. For each disease the agent has, the location of the best fit immune system substring location is computed, as well as the time until the agent will learn the disease.

11. Each agent is given an endowment of 30. The endowment varied with some experiments.

12. Each agent is given an age which is Equilikely(1, ζ).

From the initialization information, as well as an understanding of how the agents interact with one another and the landscape, and the concept of an immune system, we can construct the following agent data structure which encapsulates both the agent states and attributes.

```
struct agent{
  /* States: */
  int x;                              /* x,y position of agent       */
  int y;
  float sugar;                        /* Current resource level      */
  int immune_system[I];               /* Immune system string        */
  infection disease_list[M];          /* List of diseases agent has  */
  int disease_count;                  /* Count of active diseases    */
  int age;                            /* Agent age                   */
  /* Attributes */
  float metab;                        /* Agent  metabolism           */
  int vision;                         /* Agent  FOV parameter        */
  int maxage;                         /* Agent  life expectancy      */
  int fertile;                        /* Age when agent is fertile   */
  int infertile;                      /* Age when agent is infertile */
  char sex;                           /* Agent  sex 'm'ale/'f'emale  */
  float endowment;                    /* Starting resource level     */
  /* next agent in a linked list */
  struct agent* next;
};
```

As with agents, the concept of a disease can also be encapsulated into a data structure to enable quick processing of, and access to, the necessary information for each disease.

```
typedef struct {
  int disease[D];                     /* Disease string              */
  int position;                       /* Starting position of the    */
                                      /* best-fit substring          */
  int time_to_well;                   /* Time until agent is well    */
} infection;
```

Once all the agents are created and the landscape initialized, the agents move about and interact with the landscape and one another. For time $t = 0$ to time $t = T$ the agents do the following. In `MoveAllAgents()`, all the agents move on the landscape to the cell with the highest resource values within their FOV. All agents move before doing anything else. Although this movement is implemented as a serial process, all movement happened at the same clock tick (a pseudo-parallel approach). The agents move in sequence, starting with the head of the linked agent list.

In `HarvestResource()`, each agent collects all the resource from the cell it occupies. The cell's resource level is then set to zero.

After gathering the resource, each agent then interacts with its neighbors. The interaction may produce new agents. In `Procreate()` reproduction occurs when an agent has a neighbor of the opposite sex which is fertile, and there is an open landscape cell for the child to occupy. This open cell must be a neighboring cell of one of the parent's call. If both agents are fertile, and there is a cell for the new child agent, a child is created immediately. The child agent is initialized by selecting each attribute from one of the parents. For each attribute but one, the child agent has an equal chance to inherit from the father or the mother. The only attribute "jointly" inherited is the new agent's immune system. Since the child inherits the immune system jointly, a traversal of both parents' immune system is necessary, as follows:

```
for (i = 0; i < I; i++) {
  if (father->immune_system[i] == mother->immune_system[i])
    child->immune_system[i] = father->immune_system[i];
  else
    child->immune_system[i] = Bernoulli(0.5) ?  father->immune_system[i] :
    mother->immune_system[i];
}
```

The other agent-agent interaction occurs in `TransmitDisease()`. Disease transmission involves one disease being selected at random from the agent's diseases and passing that disease to a neighbor. If the agent has no diseases, then no transmission occurs. Otherwise, for each neighbor, a disease is selected at random from the agent's diseases. If the neighbor already has the selected disease, the transmission attempt fails. Similarly, if the neighbor is carrying the maximum number of diseases, the transmission attempt fails.

After the agent-agent interactions finish, the agent updates its states in UpdateAgents(). All states are updated except the (x, y) location which is updated in MoveAgents(). This update involves the agent consuming its metabolism's worth of resource, as well as any additional resource required for the agent's diseases, aging the agent one time unit, and updating the agent's immune system and disease list.

The agent's immune system and disease list are updated as follows. For each time step, the "time to be well" decreases by one for each active disease. When the time to be well is 0 for a disease, the agents immune system changes to be immune to that disease. The "position" variable in the disease data structure points to the starting bit location of the best fit substring in the immune system. In this way, the immune system can easily be updated to learn the disease.[6] The defeated disease, which has been learned, is then removed from the agents disease list. The agent updating can be summarized as follows

```
sim->sugar -= sim->metab;
sim->sugar -= BURDEN * sim->disease_count;
for (index = 0; index < sim->disease_count; index++) {
  sim->disease_list[index].time_to_well--;
}
RemoveDiseases(sim);
sim->age++;
```

Once the resource is consumed, procreation is finished and the immune system and disease list are updated, a mortality check is done on the agent. The mortality check is also contained in the UpdateAgent() function. The mortality check is done to see if either the agent has "starved" (run out of resource), or has lived out its lifetime. In either case, the agent "dies". When an agent dies, it is removed from the agent list. It should be noted

[6]The position is computed at the same time as the Hamming distance is, as soon as the agent is infected with the disease.

that procreation is not possible if the agent has too little resource. (Attempting procreation for an agent that will die from a lack of resource on the given turn will not produce any children.) Procreation attempts will also fail for agents who will die of old age, as the minimum old age is the maximum infertility age. However, diseases may be transmitted by a dying agent. This is not unrealistic in that some infections are still active upon death of the host.

Once all the agents have been processed, the landscape cells regrow their resource in `RegrowLand()`. Each cell's occupancy status is set during the agent movement phase. When an agent leaves a cell, the cell becomes unoccupied. When an agent moves into a cell, the cell becomes occupied. The pointer for each landscape cell makes determining the occupancy of a cell fast. To find a neighbor, one only needs to query the four neighbor landscape cells and not search the agent list.

The agents are stored in a linked list fashion. A linked list was chosen over a static structure because the agent population changes in size a great deal over time. The flexibility of a linked list to expand and contract on demand with low overhead was a primary factor in its selection. Since no sorting or ordering of any kind is necessary, more complex dynamic data structures were not considered. However, if the agent list did not change its order, the same agent would move first at each time step, would interact first on each time step, etc. So, to prevent any simulation artifacts due to the use of a list, and the synchronous-played-out-as-parallel activity of the agents, the list was shuffled after each time step in `ShuffleAgents()`. A temporary array was used to copy the pointers of the list, and the array was then shuffled using a shuffling algorithm presented in [7]. Once shuffled, the linked list was rebuilt from the shuffled pointer array.

It would be possible to use a circular linked list in this model, with a variable, random starting point. However, using a circular list would still cause problems in that if agents C and D come one after the other in the list, agent C will always move before agent D, unless agent D is the selected head of the list. Shuffling removes such artifacts. It would also be possible to use a large static circular array (because the landscape is of fixed size, there cannot be more agents than cells). With an array, shuffling is easy, or a random starting point could be used for the agents.

2.4 Time Complexity

The simulation time complexity per time step is inherently $O(A)$, where A is the current number of agents alive. As the simulation is written, there are no "individual" times for each possible event (movement, reproduction, disease transmission, etc). This leads to the question of how time conflicts are solved. As previously stated, the simulation runs in pseudo-parallel fashion — all agents move each time step, but there is a serial ordering to the movement. Thus, for movement, only one pass through the agent list is required. During the pass, each agent looks at all the cells within its FOV, selecting the cell with the largest amount of resource. The FOV search takes at most, $4 \times 6 = 24$ comparisons (4 neighboring cells, and a maximum FOV parameter of 6). Once movement is complete, a second pass through the agent list is made where each agent interacts with its neighbors, both procreating and spreading disease. For both processes, each agent considers the 4 adjacent cells, for a total of 8 possible interactions.

During procreation, the time required is constant (the fertility check takes constant time, and the birth of a child takes constant time). The spread of disease also takes constant time because, for each neighbor, one disease (if the agent is carrying any) is selected and transferred. The major time component here is searching the immune system to find the closest matching substring. This search takes $D(I - D)$ steps to find the closest substring, without any optimizations.[7]

Procreation, disease spread, the removal of diseases, the consumption of resource and the mortality checks require one single traversal of the agent list. At the end of all the interactions and required actions, the agent list is shuffled. A simple and efficient shuffle algorithm is used, which requires only one pass through the list. (It should be noted though, that two additional passes are used to move the list into an array, and then from an array back to a list). In total, five passes through the agent list are made.

2.5 Parent Model

As stated previously, the artificial society model in this dissertation is based on the model presented in *Growing Artificial Societies* (GAS) by Axtell and Epstein. In this section, I will note the major differences between the two models, and explain some of the choices I made.

I used the GAS model movement rules, as well as a similar landscape[8]. Relative to reproduction, in my model all agents are given the same endowment, not a random endowment, and their lifespan is 60-80, not 60-100. Aside from that, the reproduction models are the same for the inheritance of attributes, including the immune system.

[7]One optimization would be, from the start, only look at the 1 or 2 bits which change as one goes down the immune system string, rather than comparing all D bits for each starting position.

[8]The GAS model uses a landscape with two (non-physical) "peak" areas of resource, with declining resource in between the peaks. As discussed in section 2.3 my landscape capacity equation produces two peak areas of resource, in the same areas as the GAS model (upper right, lower left)

2.5.1 Disease Model

The GAS disease model served as an important baseline for my model. Some ideas from the GAS model were incorporated into mine, while others were discarded in favor of different approaches.

2.5.1.1 Similarities of the Disease Models

My model and the GAS disease model uses the idea of a genetic sequence to represent both the immune system and the diseases, and uses a binary string for both. Similarly, my model and the GAS disease model also uses the idea that a virus or parasite infecting an agent should extract some metabolic "fee" from the agent, that agents transmit disease among one another, and that the time for the immune system to learn a disease is the smallest Hamming distance between the disease and the immune system.

2.5.1.2 Immune System

When implementing the GAS model, as described by the authors, certain important details were ambiguous. Without a firm statement of how to implement some details where the authors were not clear, I was forced to make decisions.

The authors of the GAS model are not clear on how the immune system learns the disease. To quote, they say, "The substring in the agent immune system having the smallest Hamming distance from the disease is selected and the first bit at which it is different from the disease string is changed to match the disease." In developing my model, I took that to mean they use a "progressive learning" approach, in that at each time step, an immune string bit is flipped to match the disease string. Multiple bits may be flipped in the event of multiple infections. However, my model uses an "all or nothing" approach in that the immune system changes only when the full period of time to learn the disease has elapsed. My reasoning was that an agent will either live or die from an infection, and if the infection kills the agent, the agent's immune system would not have learned anything. The GAS authors do note that two or more diseases may share an overlapping section in an agent's immune system, and that by learning one disease, a previously learned disease could be "un-learned."

The models differ in the size of the immune system. The GAS authors chose an immune system string length of 50, while I chose 64. The GAS authors gave no direct justification for the immune system size other than that it should be significantly larger than the disease string size.

2.5.1.3 Burden

The GAS authors never directly state what the disease burden on the agent metabolism should be. In describing the concept of a disease, and justifying why it should cause an additional drain on the agent resource, they suggest, "[this] agent's metabolism would be hiked by, say, one ..." Beyond that, they give no indication of how the disease raises an agent's metabolism.

2.5.1.4 Diseases

The GAS authors chose to have a fixed set of diseases. This so-called disease list held either 10 or 25 disease, and each disease had a random length. The disease bits were drawn randomly via a coin flip. The length of the diseases in the list was a random number between one and ten. Agents then started with 4 different diseases when the list size was 10 or 10 different diseases when the disease list contained 25 diseases. There is some sound logic behind having a disease list in that one can create certain events and model the behavior. For example, the learning process of the immune system can be studied more fully with a disease list, and transmission networks can be quantified.

Although I kept the idea that each currently active disease an agent has must be unique, I chose not to use a disease list. Instead, I used random diseases of a fixed length. With a disease list, one can study how well the agents learn the diseases. However, I did not specifically study the learning of diseases or the evolution of the immune systems. The authors stated that "[with a disease list of size 10] One can track any particular disease, but the main result of this run is that society is able to rid itself of all diseases"[2].

The GAS authors are unclear on what diseases are inherited by children at birth, if any. Do they inherit diseases from their parents, or do they draw their own initial number of diseases from the disease list? In my model, the children are given their own set of initial diseases, drawn at random, and not taken from their parents. In my model, the maximum number of diseases an agent may carry come from the GAS model. In the GAS model, the disease list gives the actual number of diseases on the landscape. Thus, no agent could ever carry more diseases than the number in the list. Consequently, the maximum number of diseases is an artifact, carried over from the GAS model. However, as it will be seen in chapter 4, this artifact did not play a significant role in the model, except when determining the starting number of diseases.

2.5.1.5 Miscellaneous

The GAS authors present a very rich model with various "modules". When they introduce diseases, several other modules are also included, including agent-trade behavior. These additional modules may cause additional side effects not seen in my model. The GAS authors have a detailed set of rules which govern trade, and the agents metabolism and resource level play a role in those rules. The GAS authors also introduce a second type of resource, which is absent from my model. A more detailed discussion of the trade model is beyond the scope of this dissertation.

The GAS authors also discuss how agents transmit diseases to one another. The GAS authors discuss the "typhoid-Mary" approach to disease transmission, and a basic model of epidemiology. They give some historical justification for this, but I judged such tracking to be irrelevant in my research.

One justification for the differences between my model and the one presented by the GAS authors deals with what the model is meant to do. As the GAS authors admit, their disease list idea leads to the elimination of diseases quite easily. Because I did not want diseases to be eliminated in my model, I designed my model to ensure that diseases were always a part of the population Similarly, a large disease burden was not used because that would cause my agent population to die off. Several other minor differences were also introduced to prevent a population die-off.

Chapter 3

Experiments

The artificial society model has many system parameters, each of which can account for differing agent behavior and a corresponding stable population. I chose to do a sensitivity study of selected agent system parameters to study their effect on the stable agent population. Though they would be equally valid to study, I did not consider the landscape parameters. I selected six agent system parameters and, for each selected parameter, I varied the standard value by $\pm 50\%$.

For the standard disease model used as a baseline is where the immune system is 64 bits, the disease size is 8 bits, the initial wealth is 30.0, the disease burden is 1.0, the maximum number of diseases is 8, and the starting number of diseases is 4. In addition to those parameters, an initial population of 400 is used. Each agent has a natural life span of 60-80 time steps, and is fertile after 12 to 15 time steps. The infertility age for males is 50-60, and for females 40-50. The agent FOV parameter is between 1-6, and the metabolism between 1.0-4.0. The landscape size is 50×50 with the resource distribution as given in chapter 2.

The selected system parameters are as follows.

- Immune system string length I — The standard immune system string length was 64 bits. The values of 96 and 32 were also considered.

- Disease string length D — The standard disease string length was 8. The values of 12 and 4 were also considered.

- Initial wealth — The standard agent initial wealth was 30. The values of 45 and 15 were also considered.

- Disease burden — The standard disease burden was 1.0 unit of resource per disease per unit time. The values of 1.5 and 0.5 were also considered.

- Maximum number of diseases — The standard maximum number of diseases is 8. The values of 12 and 4 were also considered.

- Starting number of diseases K — The standard starting number of diseases is 4. The values of 6 and 2 were also considered.

In each experiment, I only changed one of the parameter values at a time, looking to quantify the parameter's effect on the agent population, and in that way, determine the sensitivity of the model to the given parameter. Thus a total of $2 \times 6 + 1 = 13$ computational experiments were run.[1]

For each experiment, 29 replications were used to produce a confidence interval for the stable agent population. For each replication, $T = 2000$ time units were simulated, with the first 1000 discarded because of the transient behavior typically present in the first few hundred time steps. The population time history for the remaining 1000 time units was

[1]Although there are $3^6 = 729$ possible combinations of experiments, I did not investigate multiple parameter changes.

averaged to give a mean population for the replication. The 29 replication means were then used to estimate the stable population size as a 95% confidence interval.

Chapter 4

Results

The experiments listed in the previous chapter provided a great deal of insight into how changes in selected system parameters can produce significant changes in population behavior. As illustrated in figure 4.1, if the population time history of a disease-free model is compared with the population time history of the standard disease model, it is clear that disease limits the size of the agent population significantly. With the standard disease model as a reference, in this chapter I study the effect of changing selected system parameters. As discussed in chapter 3 , 29 replications with $T = 2000$ were used to estimate the stable agent population. However, as in figure 4.1, and in all the other figures presented, I only show the population time history for one replication.

4.1 Initial Conditions

The respective length of the disease and immune-system binary strings is important. If the immune string is too long, there should be more immunities (by chance and inherited) and a correspondingly larger population. If the immune system string is too small, an endemic infection should occur (two or more disease continue to infect the population). Short diseases strings cause agents to gain immunity faster, which should result in a larger

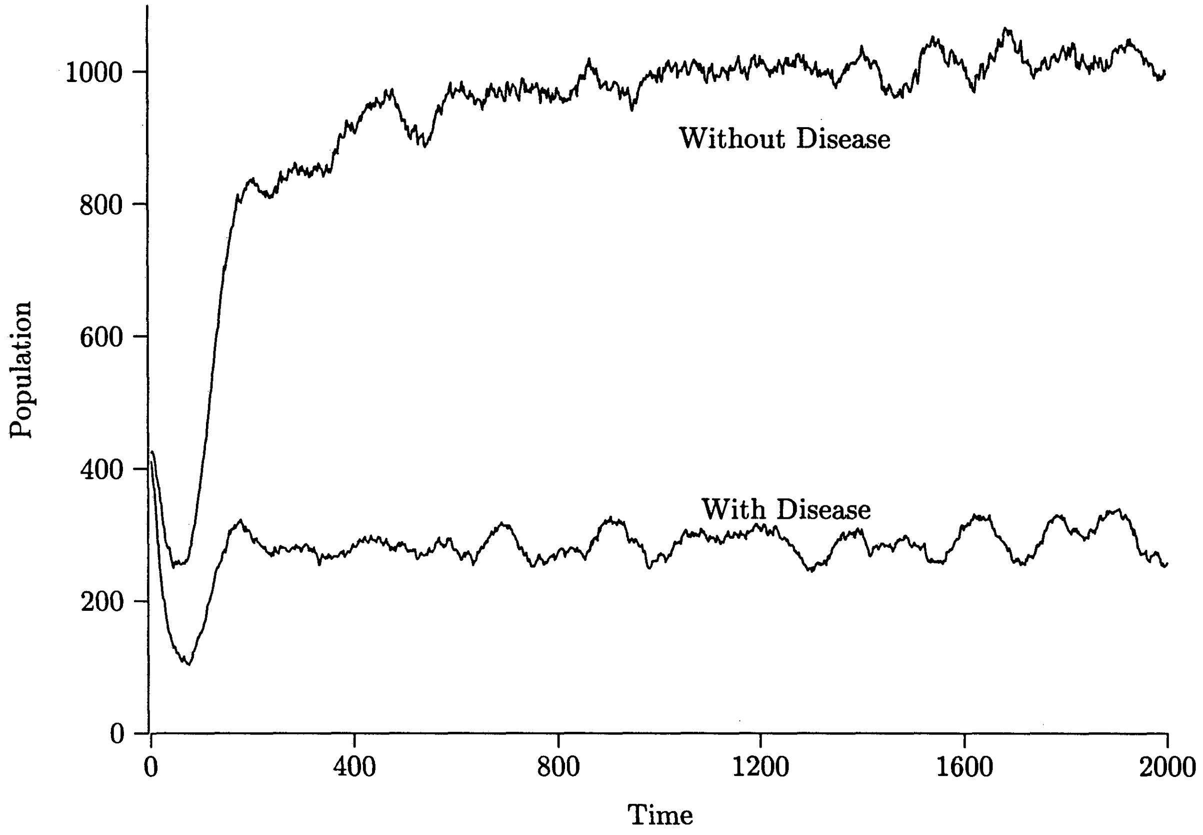

Figure 4.1: Population time history — Disease Free model vs. Standard Disease model

population. Longer disease strings should be more difficult to eradicate from a population.

I investigated disease strings of varying length, but did not find a significant difference in the stable population. While the stable population estimates differed, the results were comparable with those when the disease length was fixed at the average length. For example, if the disease string length was randomly chosen between, say, 8 and 12 (equally likely), the stable population closely resembled that produced if all the disease strings had length 10. Therefore, as discussed in chapter 2, the model assumes that all immune strings have the same size, I, and all disease strings have the same size, D.

As previously stated, when the simulation begins, the population is not fully heterogeneous. At time $t = 0$, all agents have the same number of diseases, and the same amount of resource. Each agent is given an age, selected at random between zero and the agents maximum age. This produces an age-mixed population at time zero and helps prevent a large agent die-off during the first few time steps. That is, since agents have a minimum reproduction age, if all agents started as newborns, no procreation would occur in the first 12 time steps, causing a larger population drop during those early time steps.

It should be noted that there is a characteristic population drop and spike that are found in all time histories. This transient behavior has two root causes. The first cause is childhood diseases and agents unable to find enough wealth to live longer than the first few time steps. The second cause is that all agents will die somewhere between time 0 and 80. Thus, the entire initial population will die before time 80. Together, they factors cause the large dip in the agent population during the first 100 time steps. The population spike occurs after the dip, when enough agents reach child bearing age and have enough wealth to reproduce in large numbers. Intuitively, this spike should occur after moderate amount of time has passed, as the agents are learning diseases and consuming resource at a greater rate. Once the agents have learned enough diseases, they begin to store more resource, which then allows procreation. With time, the agent population becomes heterogeneous in age, position, diseases carried, etc.

By randomizing the agents' ages, two things are hopefully accomplished — procreation is possible from time $t = 0$, and the agent population age distribution is somewhat closer to that when the population size is stable. While I did not study the age distribution (to see if it fit a sociology-expected pyramid form[1] or not), I felt that all agents being "just

[1]The "pyramid" age distribution refers to when the bulk of the population is in the younger ages brackets, and the population decreases with increasing age.

born" at $t = 0$ was not appropriate. Randomizing the agents' ages did not have a significant effect on the overall agent population. When un-randomized, the stable agent population is 279.45 ± 5.36, compared with the standard, randomized model value of 277.25 ± 5.07.

4.2 Immune System Size

A larger immune system should provide for more immunities, by chance, but also for better partial-matches of diseases, also by chance. Thus, an agent with a larger immune system should have a higher chance of survival, resulting in a larger stable population. An agent with a smaller immune system should have more overlap between the diseases, where two or more diseases have a best fit pattern which overlaps — the same bits in the immune system partially code for both diseases. This overlap was expected to cause an endemic infection — once an agent becomes immune to one disease, the agent is immediately susceptible to the other. However, this "endemic infection" scenario is not observed. Figure 4.2 illustrates how the agent population varies, and it appears that both larger and smaller immune systems do not have a significant effect on the overall population of the agents.

Figure 4.2 was produced by gathering the agent population at each of the first 2000 time steps for three different immune system sizes. Between each run, only the agent immune system size I was varied. The stable population values for the increased and the decreased immune system have a similar value 260.39 ± 10.90 and 274.49 ± 4.99, respectively, which are different from the standard model at 277.25 ± 5.07, but not significantly. These data show the immune system to be a rather robust parameter.

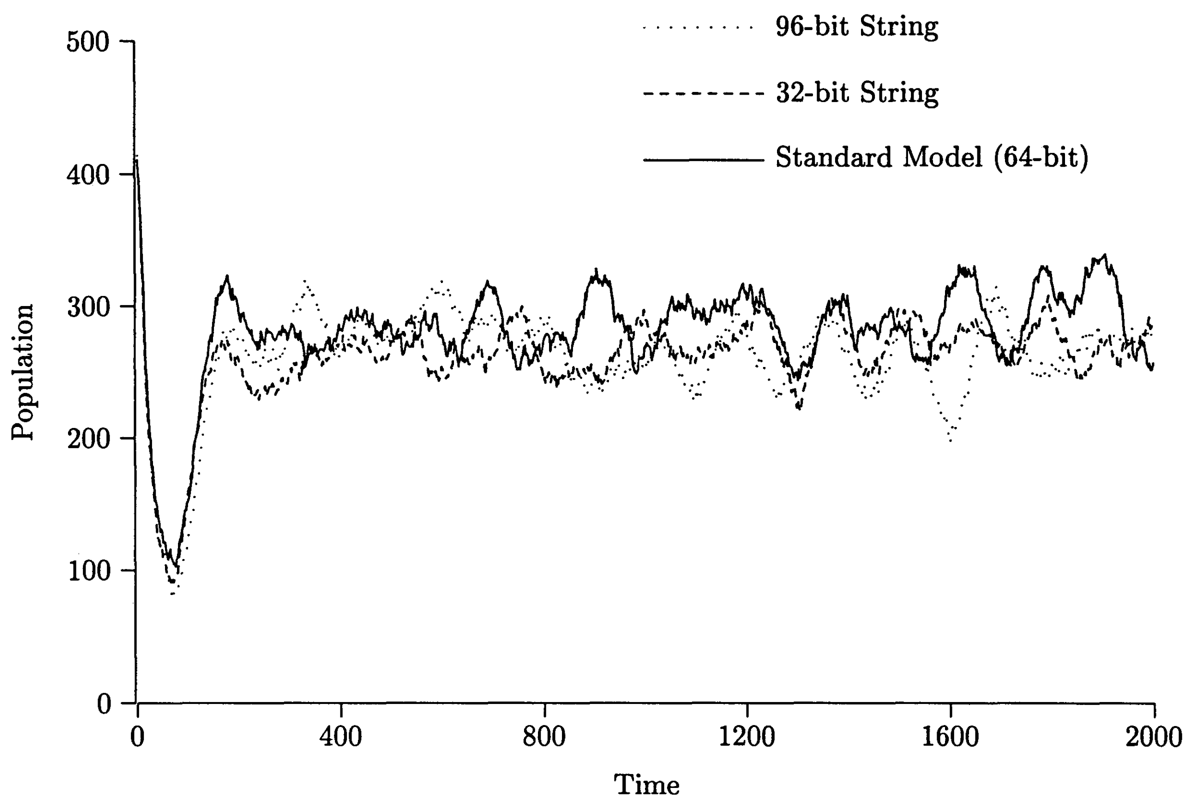

Figure 4.2: Population time history — three immune system sizes, I

4.3 Disease Size

A disease becomes more "complex" as the disease string length increases. A longer disease should result in more agent deaths because the agent's immune system will take a longer time to learn the disease, which in turn should result in a smaller stable population. Conversely, a shorter disease string should result in a larger stable population, since an agent will learn the disease quicker. As illustrated in figure 4.3, a longer disease size does produce a smaller agent population. The stable population is 210.64 ± 8.42. However, while the shorter disease size does produce a marginally significant change, with a stable population of 260.76 ± 3.65, this change is not as pronounced as the increased disease length, and it is counter intuitive. For reference, the standard model stable population is 277.25 ± 5.07.

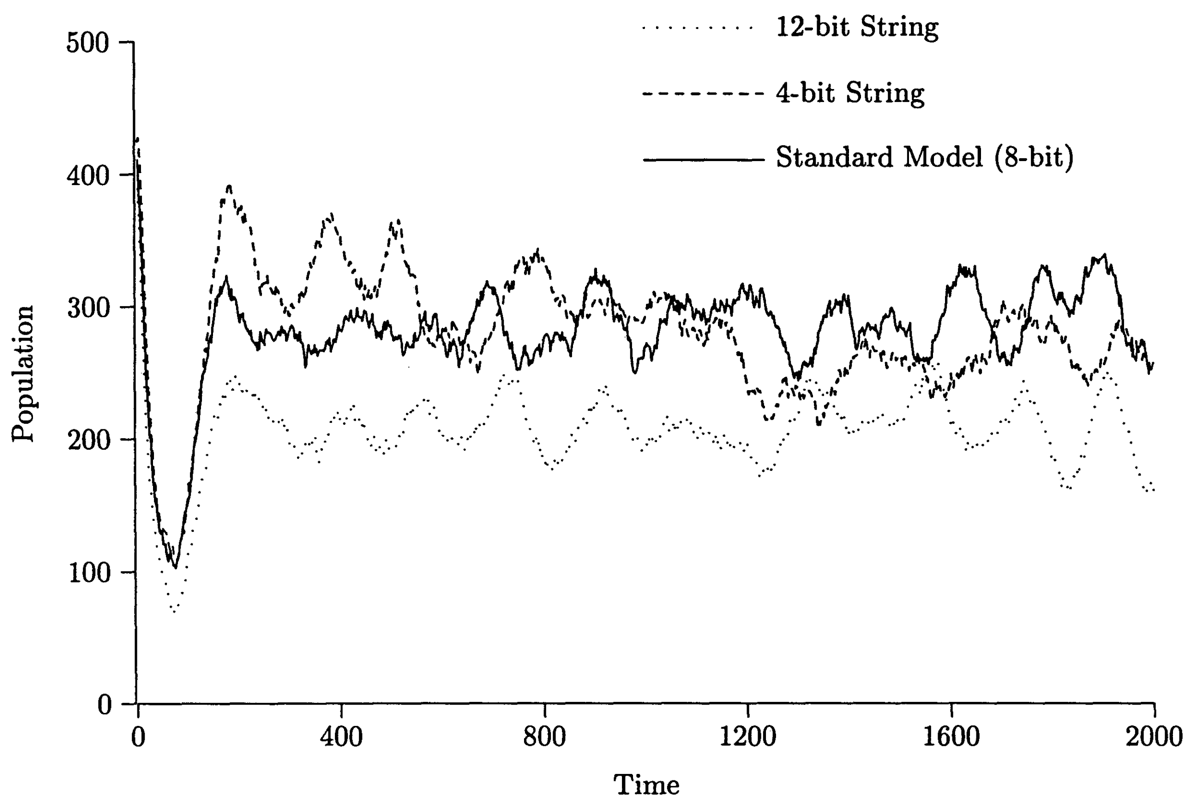

Figure 4.3: Population time history — three disease sizes

4.4 Initial Wealth (Endowment)

When an agent is given a larger initial wealth, the expected outcome is for the agent to live for a longer period of time. That is, the agent should be able to survive the childhood disease period and any short-term lack of resource problems. However, a higher initial wealth also makes procreation more difficult, as an agent must have at least as much as its endowment to reproduce. Thus an agent may live longer, but the agent will have to gather more resource before being able to procreate. The opposite can be said for a lower initial wealth. An agent with a lower initial wealth may not survive the childhood disease period, but the agent will be able to procreate at a much greater rate. As figure 4.4 shows, we see the expected behavior — a larger initial endowment produces a lower stable population, and a smaller initial endowment produces a higher stable population. The lower initial wealth produces a large population increase over the standard model, with a stable population of 359.12 ± 3.78. Similarly, the increased initial wealth produces a large population decrease over the standard model with a stable population of 197.36 ± 15.19. For reference, the standard model stable population is 277.25 ± 5.07.

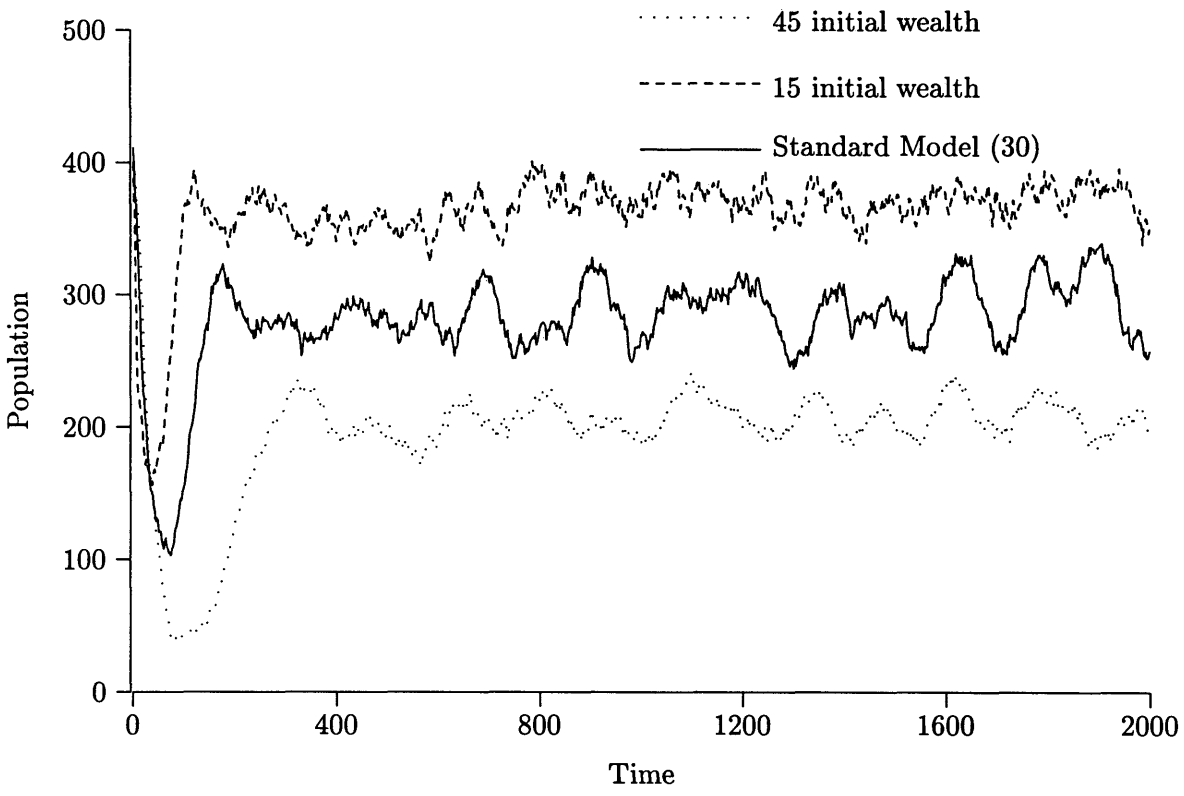

Figure 4.4: Population time history — three initial wealths

4.5 Burden

The parameter which has the most impact on the agents is the burden. An agent may have many diseases, but if the diseases are not placing a significant drain on the agent's resources, then the diseases have a very limited effect on the agent. Figure 4.5 illustrates that the burden does have a significant effect on the agent population, as expected. When the burden is lowered, the agent population reaches a much higher stable value. Conversely, when the burden is raised, the population reaches a lower than normal stable value. A stable population difference of about 100 agents exists in each case. The lower burden stable population is 417.82 ± 3.27, whereas the higher burden stable population is 200.54 ± 4.87. The standard model falls between the two, with a stable population of 277.25 ± 5.07.

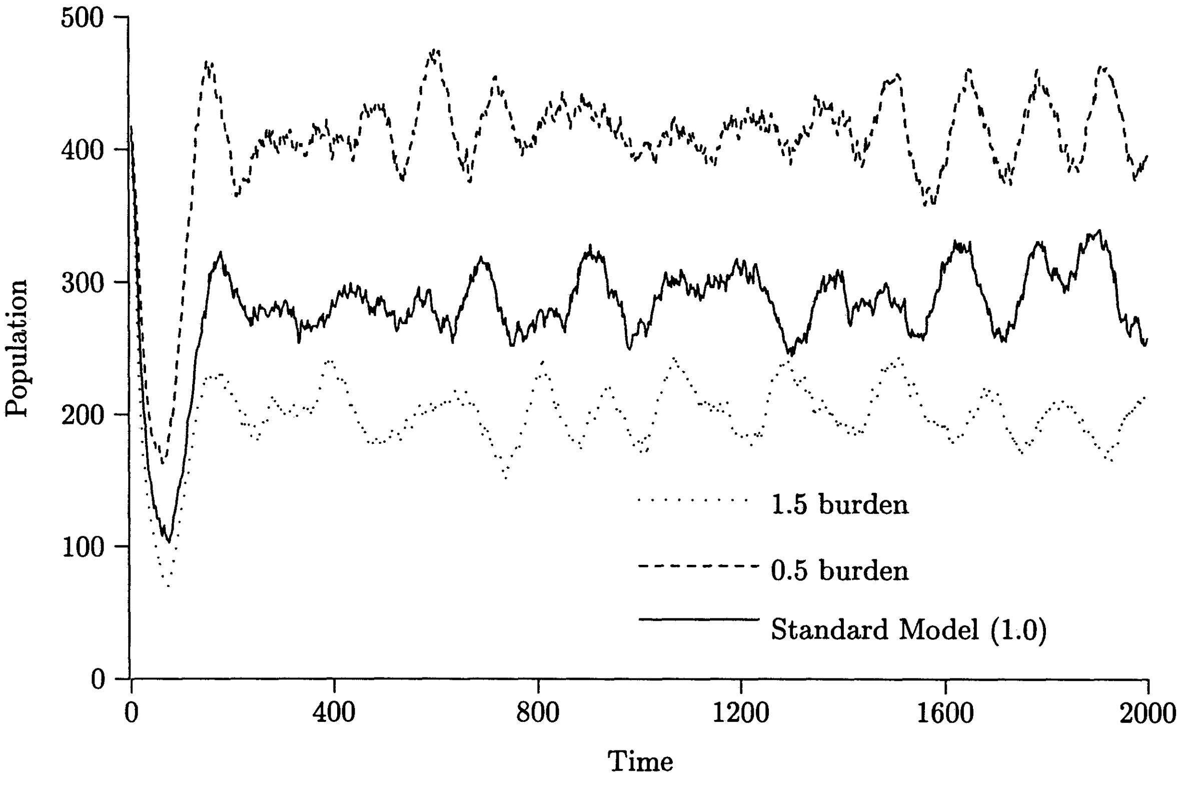

Figure 4.5: Population time history — three disease burdens

4.6 Max Diseases

If an agent can be infected with more diseases then we should find a lower agent population. Since an agent can carry more diseases, the agent is more likely to die from one of those diseases and more likely to transmit and spread multiple diseases. Both of these possibilities lend themselves to a lower population. Conversely, if an agent can only be infected with less diseases, then a higher population should be expected. Since an agent can carry less diseases, a smaller strain is placed on the agent. Carrying less diseases also lends itself to eradication of disease. Since less diseases are carried, fewer can be transmitted, and thus disease can be eradicated. However, as figure 4.6 shows, the maximum number of diseases is not a significant parameter. The decreased maximum number of diseases provides an insignificant difference in the stable populations, settling at 274.70 ± 6.28, whereas the standard model and the increased number both bore the same result of 277.25 ± 5.07.

These results show that no agents ever contract the maximum number of diseases for the standard model, let alone my modified example here. Such an event is a good thing because limiting the number of diseases an agent can carry is an artificial condition. By showing that increasing this artificial maximum does not change the results, I conclude that the maximum used during the other experiments was valid (other than the increased starting diseases experiment)[2].

4.7 Starting diseases

If an agent starts with fewer diseases, we have a situation similar to when an agent can carry a smaller maximum number of diseases. The eradication of diseases is more likely in this case. Since fewer diseases are created to start with, agents can be expected to rid themselves of all disease. Also, since agents start with fewer diseases, the transmission of disease becomes less likely as well. Overall, having fewer starting diseases should result in a

[2]When the starting number of diseases is increased, the agent carries 6 diseases, which is close to the maximum number of diseases. In this case, I would expect that the maximum number of diseases may be reached for multiple agents. In such a case, the maximum number of diseases would inhibit the disease model

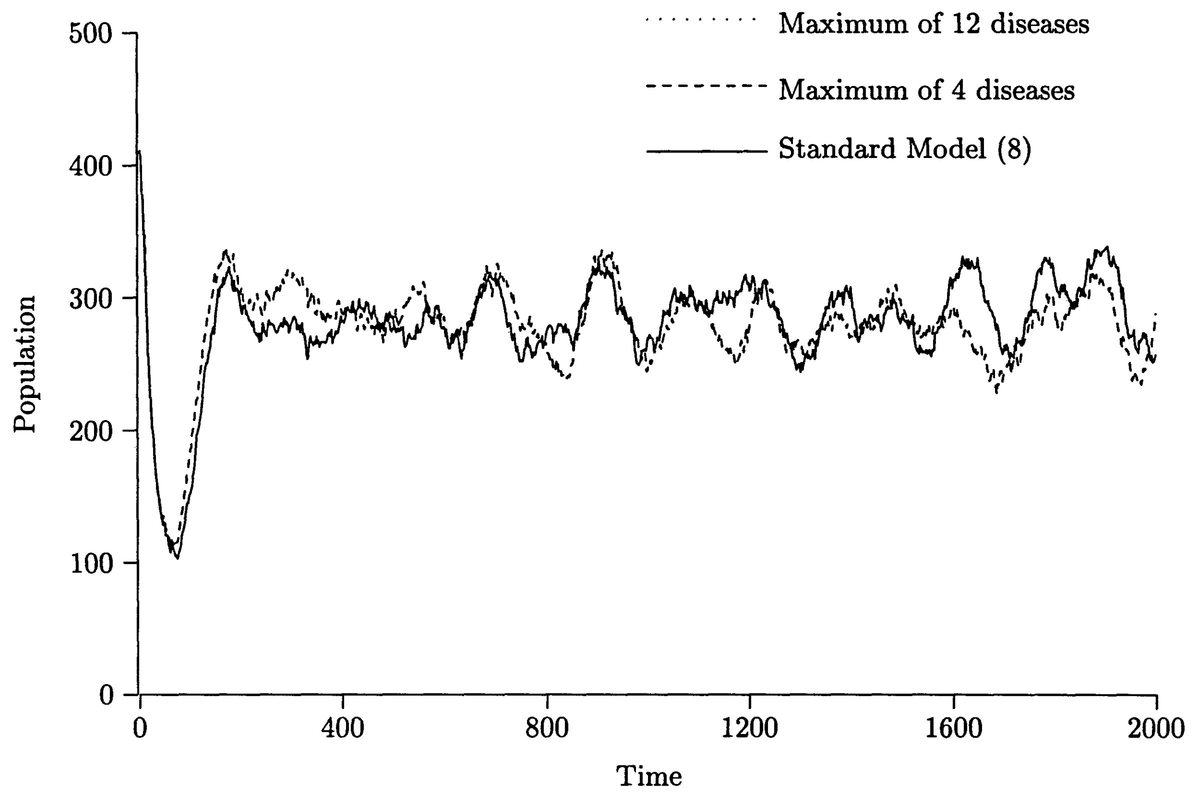

Figure 4.6: Population time history — three maximums of diseases carried

higher population of agents. Conversely, if agents start with more diseases, we should expect more agents to die off from the childhood disease phase, and those who do live to adulthood should still be contagious enough to pass on their diseases. This should result in an overall lower stable agent population. However, as illustrated in figure 4.7, we see that decreasing the starting number of diseases does not significantly change the stable population, nor does increasing the starting number of diseases produce a significant effect on the stable population. Increasing the starting number of diseases yields a stable population of 272.15 $\pm$ 5.27. The stable population for a decreased number of starting diseases is 282.34 $\pm$ 4.83. For reference, the standard model's stable population is 277.25 $\pm$ 5.07.

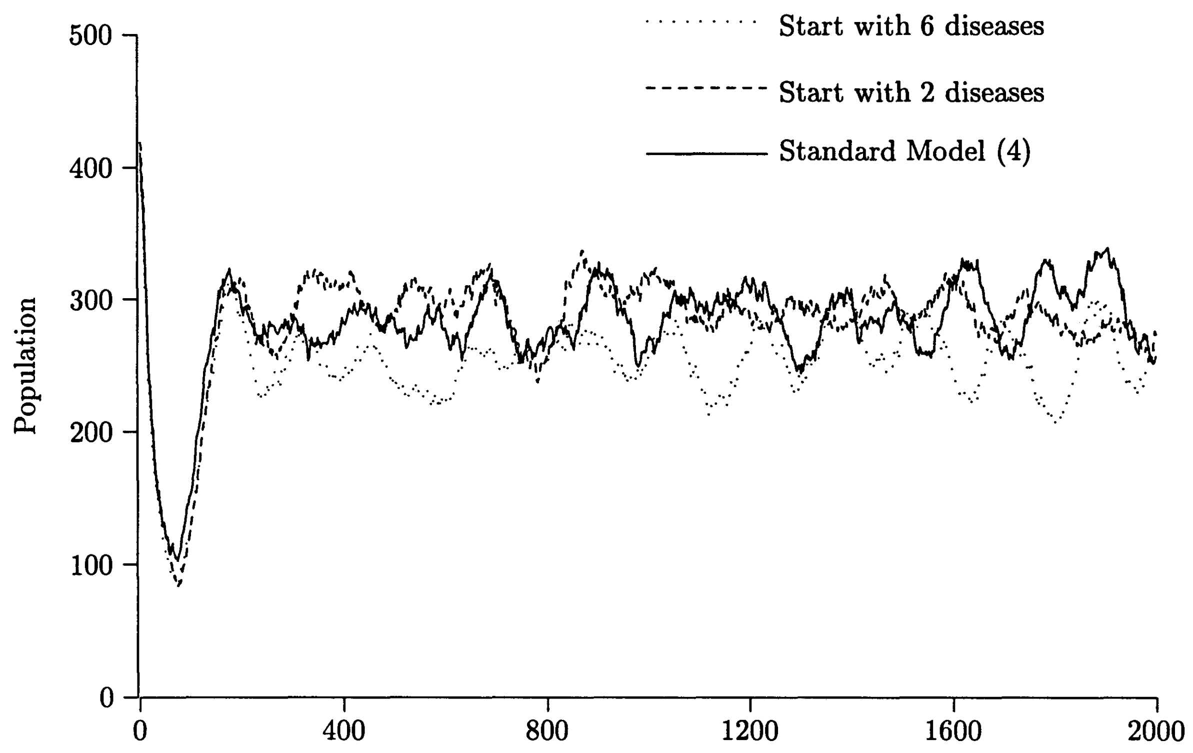

Figure 4.7: Population time history — three numbers of starting diseases

4.8 Summary of Results

As seen in some of the experiments, the expected and intuitive changes in the stable population did not necessarily happen. While the graphs present only one replication of the population time history, table 4.1 shows the stable population as an interval estimate based upon all 29 replications.

Parameter	Decreased	Standard	Increased
Disease Burden	417.82 ± 3.27	277.25 ± 5.07	200.54 ± 4.87
Initial Wealth	359.12 ± 3.78	277.25 ± 5.07	197.36 ± 15.19
Disease Size	260.76 ± 3.65	277.25 ± 5.07	210.64 ± 8.42
Starting Number of Diseases	282.34 ± 4.83	277.25 ± 5.07	272.15 ± 5.27
Immune System	274.49 ± 4.99	277.25 ± 5.07	260.39 ± 10.90
Max Number of Diseases	274.70 ± 6.28	277.25 ± 5.07	277.25 ± 5.07

Table 4.1: The summarized stable populations and their confidence intervals

4.9 Interesting Results

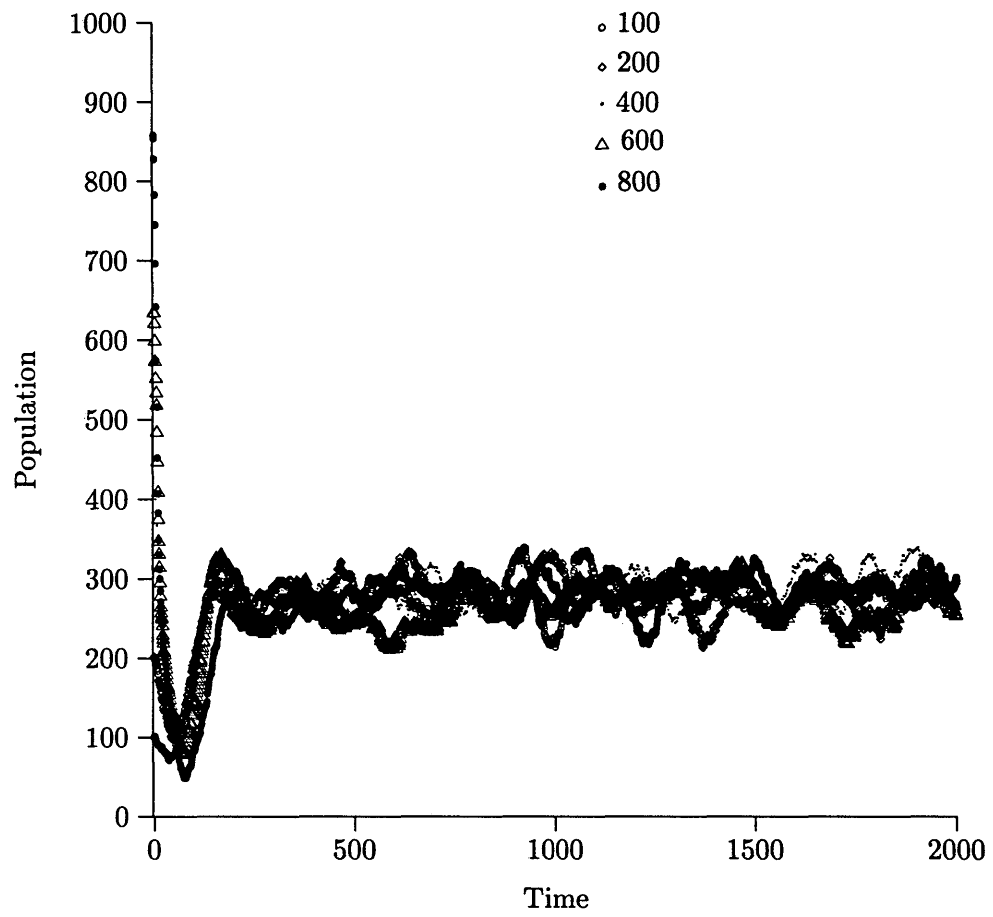

Figure 4.8: Population time history in the standard model, for varying initial populations

A very interesting result I found is that essentially the same stable population value is reached, regardless of the initial population, for the standard disease model. If I increased the initial population I found approximately the same stable population as detailed in figure 4.8 and table 4.2. Similarly, if I lowered the starting population, up to a point, I found the same stable population values as well.[3] I do make the assumption that if there is a stable population for the standard model, there must also be a stable population for the

[3]There is a "minimum" possible initial population. Initially there must be enough agents to survive the initial deaths, and being able to reproduce, raising the agent population from there to the stable level.

experimental models.

Initial population	100	200	400	600	800
Stable population	266.42 ± 7.21	269.87 ± 6.74	277.25 ± 5.07	273.60 ± 6.52	276.72 ± 5.19

Table 4.2: Stable population values for varying initial population, replications

While still fine tuning the model, I found another interesting result. When the agent's maximum age was increased, the stable population tended to be lower, though not by a large amount. As agents live longer, they no longer procreate. They continue to consume and gather resource, but do not contribute to the agent population. Since there is no "inheritance" in the model, the resources these older agents collect are "taken with them" when they die. This population decrease can be seen graphically in figure 4.9. The stable population for the longer-aged model is 242.01 ± 20.01.

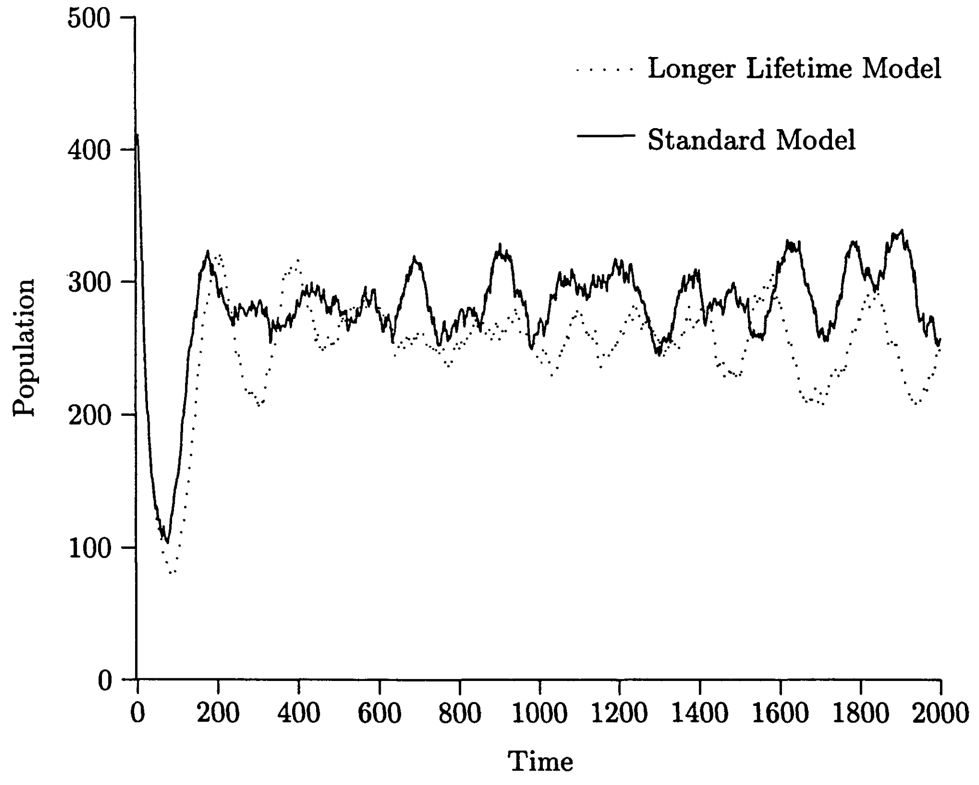

Figure 4.9: Population time history where agents may live longer

4.10 Experimental Orderings

As noted in chapter 2, there is an ordering of the events in my model. First, all agents move and gather resource, then each agent attempts procreation, transmits disease, consumes resource, and updates its immune system. After that, the landscape regrows resource, and time is advanced. This ordering of the events is somewhat arbitrary, and it can be shown that some different orderings do produce different results. While this is not a desired behavior, it does happen, and warrants further study. As discussed in chapter 2, the standard model is arranged as:

```
1 InitializeLandscape()
2 InitializeAgents()
  For t = 0 to T
3   MoveAllAgents()
    For each agent
4     HarvestResource()
5     Procreate()
6     TransmitDiseases()
7     UpdateAgents()
    End For
8   RegrowLand()
9   ShuffleAgents()
  End For
```

The ordering of lines 8 and 9 may be swapped without a change in the results. The initialization lines 1 and 2 must be in that order. That is, the landscape must be initialized before placing agents on it, otherwise the agent positions would be overwritten. Lines 5 and 6 rely on neighbors, and both will change agent states (5 may change the agent's wealth, and 6 may change the agents disease list). However, since children do not inherit diseases from their parents, and a newly acquired disease does not drain any resource until line 7, lines 5 and 6 are interchangable.

It is possible to remove the inner for loop, and do lines 3, 4, 5, 6, 7 on a single agent basis. This would eliminate the `MoveAllAgents()` function, and put a `MoveAgent()` function inside the `For each agent` loop. However, as discussed below, this would be a fundamental change of the model.

If we change the behavior of the inner loop to the order $(4, 7, 5, 6)$ we find very different results. Since in this case the agent consumes resource, and removes diseases which have been learned before doing anything else, a significant drop in the number of infected agents occurs. This drop in the number of infected agents then causes a population boom, resulting in a very large stable population of 925.60 ± 8.12. Since the agents remove diseases they conquer before passing them on, most agents will remove a disease within 1 or 2 turns. The end result of this ordering is that fewer diseases are spread, and so the number of active diseases in the agent population dwindles. With fewer diseases circulating, most agents will be healthy, and thus have more resource. More resource in turn results in more procreation, and thus a larger population. Certainly the newly created agents are given their own set of diseases, but the children learn the diseases quickly enough to avoid significantly transmitting the diseases.

If the inner for loop is removed, so that steps 3, 4, 5, 6, 7 are done for each agent, different results are also obtained. A very important point here is that this removes the "parallel" movement of agents. This is a fundamental change in the model. The major difference is that one agent may be in the current "time step", having just moved. However, this agent may interact with agents in the "past time step". Such agents in the past have not yet moved or done anything else. As the model is described in chapter 2, no agents interact until they have all moved. Moving all the agents first synchronizes the time before any agent-agent interactions happen. The time discordance created by removing the inner for loop is responsible for the differences in the results.[4] This change resulted in an stable population of 345.05 ± 2.02, which is a significantly different from the standard model's stable population of 277.25 ± 5.07.

4.11 Consistency Checks

I can claim confidence in my results for the following reasons. While running the simulation, I tracked several variables in addition to the current population size, such as the number of agents with active infections, and the number of agents who have no infections.

- Intuitively, the total population should be the number of agents with no infections plus the number of agents with infections. This was the case for my simulation.
- I also tracked the birth/death statistics. At some time t, the population should be the population at time $t - 1$ plus the number of births minus the number of deaths. Again, this was the case for my simulation.
- Similarly, I also ran my simulation with no diseases (no immune system, no disease size, no maximum number of diseases and no starting diseases) to have a "baseline" stable population for the landscape. As one would expect, the population of agents when there are no diseases is significantly higher than the population of the disease models (see figure 4.1).

[4]One possible explanation is that if two agents have moved away from the areas of higher resource, their net-gain in resource for one or two time steps may be negative. If we allow the agents to interact in an unsynchronized manner, this negative imapact may not have cought up with the agent yet.

Chapter 5

Summary and Conclusions

The model presented in this dissertation is rich with parameters. While my experiments involved changing one parameter at a time, I would not be surprised if it were possible to modify a mixture of the parameters to yield desired contrived results.

Some of my experimental results were counter intuitive. In particular, the immune system changes did not produce the expected results. I expected a longer immune system string to provide more immunities, and thus a larger stable population. This was not the case.

The burden and initial wealth are the two most sensitive parameters in my model. The burden's effect on the model is fairly straightforward. Since the only effect a disease has on an agent is to raise its metabolism, the burden should be a major factor in determining the stable population. The initial wealth's role is less straightforward. While it plays no explicit role in the disease transmission model, the initial wealth does play a direct role in the reproduction model. A small initial wealth will result in greater procreation, since a parent agent need not gather as much resource before being able to reproduce. However, one would also expect a small initial wealth to limit the population because more child agents should die from a lack of resource.

I am rather disturbed that changing the ordering of the model can produce drastically different results. Accordingly, I think this issue should be investigated in more detail. Although the synchronous approach I took does work, an event-based approach would be preferable, in my opinion. If movement, resource consumption, and agent-agent interactions were all individual events with their own "times", some of the problems I found would not occur.

Switching to individual event times should be possible. This change is presented in [8]. Such a change would add a good deal of complexity to the model, but would also serve to remove some of the simulation artifacts the GAS authors introduce. Since the agents consume resource at a constant rate, one can always tell, for a given amount of time, how much the agent consumes. Similarly, the landscape regrows resource at a constant rate, so the level of resource at a cell can be determined if we know the last time the cell was harvested. The time it would take for an agent to learn a disease is known as soon as the agent contracts the disease. So all that is left is movement and agent-agent interactions. If we pick a constant rate for the movement, and the interactions, then this model could be made into an event driven model. The shuffling of the agent list would not be necessary in an event driven model.

A circular immune system would also be a nice change to the model. As presented in this dissertation, the immune system has a start and an end. With a circular immune system this would not be the case. While I would not expect any "major" changes to occur, the use of a circular immune system would allow agents to better utilize the first few and last few bits of their immune system.

Appendix A

Stochastic Functions

The following stochastic functions and excerpt from libraries `rngs` and `rvgs` were written by Steve Park and David Geyer [7]. The function `Random()` uses a Lehmer random number generator to produce a floating point value between 0.0 and 1.0. The function `PutSeed()` is used to initialize the generator. Each stochastic component in the computational model is allocated one of 256 disjoint random streams. A call to the function `SelectStream()` defines the active stream for the next random number to be generated. This allows for a unique source of randomness for each component [7]. The remaining functions are stochastic distributions whose mean and standard deviation is summarized in Table A.1.

```
#define MODULUS    2147483647
#define MULTIPLIER 48271
#define CHECK      399268537
#define STREAMS    256        /* # of streams     */
#define A256       22925      /* jump multiplier */
#define DEFAULT    123456789  /* initial seed     */

static long seed[STREAMS] = {DEFAULT}; /* current state of each stream   */
static int  stream        = 0;         /* stream index, 0 is the default */
static int  initialized   = 0;         /* test for stream initialization */
```

```
  double Random(void)
/* ================================================================
 * Random returns a pseudo-random real number uniformly distributed
 * between 0.0 and 1.0.
 * ================================================================
 */
{
  const long Q = MODULUS / MULTIPLIER;
  const long R = MODULUS % MULTIPLIER;
        long t;

  t = MULTIPLIER * (seed[stream] % Q) - R * (seed[stream] / Q);
  if (t > 0)
    seed[stream] = t;
  else
    seed[stream] = t + MODULUS;
  return ((double) seed[stream] / MODULUS);
}

  void PutSeed(long x)
/* ===============================================================
 * Use this function to set the state of the current random number
 * generator stream according to the following conventions:
 *    if x > 0 then x is the state (unless too large)
 *    if x < 0 then the state is obtained from the system clock
 *    if x = 0 then the state is to be supplied interactively
 * ===============================================================
 */
{
  char ok = 0;

  if (x > 0)
    x = x % MODULUS;                    /* correct if x is too large  */
  if (x < 0)
    x = ((unsigned long) time((time_t *) NULL)) % MODULUS;
  if (x == 0)
    while (!ok) {
      printf("\nEnter a positive integer seed (9 digits or less) >> ");
      scanf("%ld", &x);
      ok = (0 < x) && (x < MODULUS);
      if (!ok)
        printf("\nInput out of range ... try again\n");
    }
  seed[stream] = x;
}

  void SelectStream(int index)
/* =================================================================
 * Use this function to set the current random number generator
 * stream -- that stream from which the next random number will come.
 * =================================================================
 */
{
  stream = ((unsigned int) index) % STREAMS;
  if ((initialized == 0) && (stream != 0))   /* protect against        */
    PlantSeeds(DEFAULT);                     /* un-initialized streams */
}

   long Equilikely(long a, long b)
/* ==================================================================
 * Returns an equilikely distributed integer between a and b inclusive.
 * NOTE: use a < b
 * ==================================================================
 */
{
  return (a + (long) ((b - a + 1) * Random()));
}

   long Bernoulli(double p)
/* ========================================================
 * Returns 1 with probability p or 0 with probability 1 - p.
 * NOTE: use 0.0 < p < 1.0
 * ========================================================
 */
{
  return ((Random() < (1.0 - p)) ? 0 : 1);
}



   double Uniform(double a, double b)
/* ===========================================================
 * Returns a uniformly distributed real number between a and b.
 * NOTE: use a < b
 * ===========================================================
 */
{
  return (a + (b - a) * Random());
}
```

Distribution	**Mean**	**Standard Deviation**
`Equilikely(a,b)`	$\frac{a+b}{2}$	$\sqrt{\frac{(b-a+1)^2-1}{12}}$
`Bernoulli(p)`	p	$\sqrt{p(1-p)}$
`Uniform(a,b)`	$\frac{a+b}{2}$	$\frac{b-a}{\sqrt{12}}$

Table A.1: Stochastic Function Descriptions

Bibliography

[1] JEFFREY YOUNG. *Using Computer Models to Study the Complexities of Human Society*, Chronicle of Higher Education 1998

[2] ROBERT AXTELL, JOSHUA EPSTEIN. *Growing Artificial Societies: Social Science from the Bottom Up* 1996, MIT Press.

[3] NIGEL GILBERT. in Social Research Update, Editor, University of Surrey, UK, 1993, Issue 6

[4] ANDREAS FLACHE, RAINER HEGSELMANN in Journal of Artificial Societies and Social Simulation Vol 1 No. 3, University of Surrey, UK, 1998

[5] MICHAEL MACY in Journal of Artificial Societies and Social Simulation Vol 1 No. 1, University of Surrey, UK, 1998

[6] FEDERICO CECCONI, DOMENICO PARISI in Journal of Artificial Societies and Social Simulation Vol 1 No. 2, University of Surrey, UK, 1998

[7] STEVE PARK, LAWRENCE LEEMIS Discrete-Event Simulation: A First Course unpublished, 1994

[8] BARRY LAWSON Asynchronous Time Evolution in Artificial Society Models PhD Dissertation Proposal, College of William & Mary, 1998.

VITA

Nathan Thomas Moore was born and raised in Springfield, Virginia. He graduated from Annandale High School in Annandale, Virginia, in June of 1994. Nathan received his Bachelor of Science from the College of William and Mary in 1997, concentrating in Computer Science. Nathan is currently a Masters of Science candidate at the College of William and Mary, studying Computer Science. Nathan has completed all of his required course work, and is currently finishing his thesis, expecting to graduate in December of 1999. Nathan has been accepted to the PhD program in computer science at Dartmouth College, where he will attend in the Fall of 1999.