Introduction to Agent-Based Modeling

Introduction to Agent-Based Modeling

WITH APPLICATIONS TO SOCIAL, ECOLOGICAL AND SOCIAL-ECOLOGICAL SYSTEMS

MARCOJANSSEN

Contents

Preface

This textbook aims to teach upper-level undergraduate students and graduate-level students in the life and social sciences the basics of agent-based modeling. It is based on teaching regular graduate and undergraduate courses at Arizona State University. The first version of the book was published in 2010 as a textbook for undergraduate students. The current version is an extended version with more advanced topics of interest to those who want to use agent-based modeling as a research tool.

The book introduces you to the method of agent-based modeling, but will also use these models to discuss general concepts in social science and ecology. At the end of the book, you will have a basic understanding of various concepts of complex adaptive systems, the principles of agent-based models, and have derived basic experience with programming in NetLogo, the software tool we use for agent-based modeling in this book. The version we use in this book is 7.0.4.

This ebook includes links to additional material, such as Wikipedia pages that include a more detailed discussion of certain concepts, or Youtube videos of lectures. We also provide the NetLogo files of the models that we use in this book. We will purposely use examples in the NetLogo library to make use of the directly available models. Still, we also discuss various examples that are specially developed for this book. Those models are available in the COMSES Computational Model Library, and we will

provide links to them in the text. We expect the readers will explore the links if they are not familiar with the concepts and download the models to explore them when reading the book.

The first three parts of the book provide an introduction to the various methods and concepts, and the parts after that contain more specialized topics in ecology and social science. The first Part of the book introduces complex adaptive systems and modeling concepts, and we will get our first introduction to NetLogo. In Part 2, we will discuss a series of classic agent-based models to extend your understanding of concepts of complex adaptive systems as well as derive more experience with the NetLogo. In Part 3, we will discuss various methods and tools to analyze agent-based models systematically. This will be required if we want to use agent-based modeling as a tool for scientific research.

Part 4 of the book focuses on ecosystem management and discusses simple as well as comprehensive models of population dynamics, resource governance, and rangeland management. In Part 5, we focus on the social sciences, especially on the decision making of agents who are connected via social networks.

The version of the book you have in front of you is incomplete. More chapters and extensions of existing chapters are in the pipeline, and I will update the book once or twice a year. Since you have bought this book as an ebook, you will get a new version when they are available. They will also be announced at https://intro2abm.com/.

This book complements some other textbooks, such as Agent-based and Individual-Based Modeling by Steven Railsback and Volker Grimm, which is more ecology focused, and An Introduction to Agent-Based Modeling by Uri Wilensky and William Rand, which is more focused on the Netlogo software and its applications.

In developing this book, I have received feedback from many people. First, I like to thank my students for taking my agent-based modeling courses over the years. The saying is that the best way to learn is to teach, and I learned a lot from my students. I also

like to thank Bing-Bing Zhou, Ilkhom Soliev, Nathan Rollins, Jennifer Fraser, and Alicia Tenza Peral for feedback on earlier versions of the book. If you have any feedback, don't hesitate to contact me at Marco.Janssen@asu.edu to continuously improve and extend the material.

PART I

TOOLS AND CONCEPTS

This part of the book introduces basic concepts used in the rest of the book like complex adaptive systems, and emergence. We will also introduce the basics of programming and the programming software we will use in this book: NetLogo.

CHAPTER 1

Complex Adaptive Systems

Key Concepts

By the end of this chapter. you will

1. Know the definitions of complex adaptive systems, emergence, and stigmergy
2. Know examples of these concepts

A little ant is walking on the sandy soil of Arizona looking for food. The food is meant for feeding the brood that is being taken care of by other ants in the colony. Different ants have different tasks in the colony, and the ant we are following is charged with leaving the nest every day in search of food. The amazing aspect of ant colonies is that such a complex organizational structure exists that is not controlled by the queen or a small group of bureaucratic ants. There is also no plan or a to-do list that the ants are following. No, the complexity of the ant colony emerges out of the local interactions among the ants. The ant we observe follows a trail of pheromones, indicating that other ants of the colony have found food nearby and dropped the pheromones on their way back to the nest. Therefore, the use of pheromones is a way in which ants communicate to others, “follow my trail, and you may find food.”

The pheromone trail will evaporate at a certain speed, and will therefore only be of limited use. If other ants do not follow this trail, bring back food, and add their own pheromones to the trail, the trail will disappear. But when successful use by others enhances the trail, a highway of ants may emerge. On such a highway, we see one lane of ants empty-handed following the pheromone signal, and the other lane of ants bringing food back to the nest.

There is an enormous diversity of ant species, all of which have variations in their social organization. Some ant-species produce ants with different physical characteristics that distinguish their role in the colony: workers, foragers, soldiers, etc. In other ant-species, all of the colony's ants are physically similar and can switch roles when needed. For example, when an ant-eater kills foraging ants, the colony will experience a reduction of food being delivered, which then signals other ants to exchange their caring for the brood role for a foraging role.

Ants change the environment, which subsequently changes the behavior of other ants. This indirect influence of agents via the change of the environment is called stigmergy. Another example of stigmergy is the digital trials we develop when we interact with websites. By buying books or listening to music songs, we leave information "pheromones." We get recommendations about other books or music that people "like you" also bought or listened to. These stigmergic interactions can lead to a reinforcement of choices. Popular movies featured on YouTube tend to get even more viewers. If we want to understand how certain books, movies, or songs become so popular, we need to look into the various ways choices are influenced and reinforced by others.

Emergence

Emergence is a macro-level phenomenon, like the functioning of an ant colony, resulting from local-level interactions. Like the cartoon in Figure 1, understanding macro-level phenomena can only be derived by studying the consequences of micro-level agents' behavior. Micro-level agents make decisions based on local

cues and do not know the macro-level implications of individual behavior.

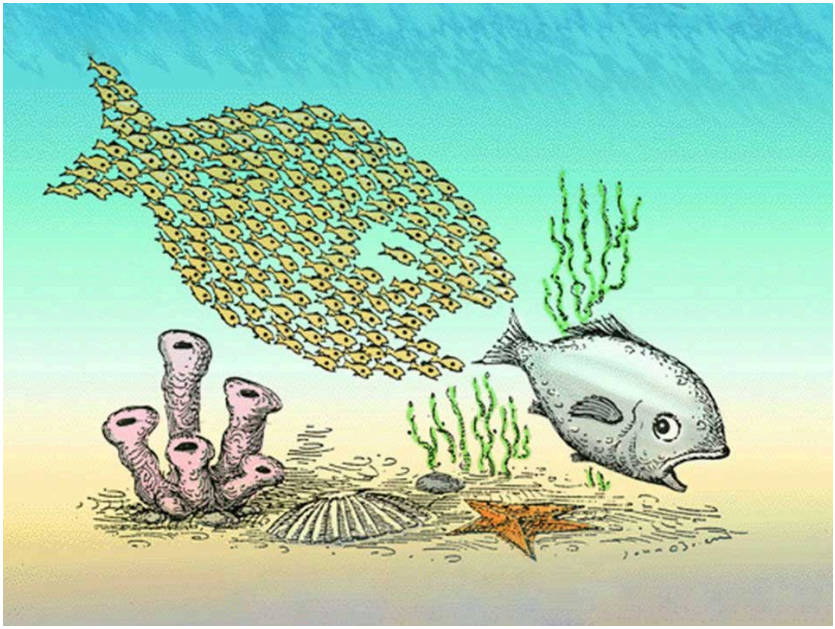

Figure 1. Macro-level phenomena emerge from the behavior of agents on the micro-level.

Other examples of emergence are:

Path Formation

Paved paths are not always the most desirable routes going from point A to point B. This may lead pedestrians to take short cuts. Initially, pedestrians walk over green grass. Subsequent people tend to use the stamped grass path instead of the pristine lawn, and after many pedestrians, an unpaved path is formed without any top-down design.

Look around your campus, and you will find examples of paths that have emerged from a bottom-up design. In a later chapter, we present how to model this phenomenon.

Traffic Jams

Have you ever been stuck in a traffic jam only to reach the front of the slow-down to discover that there was no apparent reason for it? Sometimes, traffic jams, defined by slow-moving cars, develop rapidly in the opposite direction of traffic flow. As a consequence, you may enter a traffic jam that was formed many miles ahead. Traffic jams can also develop without direct causes like accidents. When drivers have different preferred speed levels, they need to slow down as they approach a slower car in front of them. As a consequence, other cars behind this car will need to slow down too. As we will see in later chapters, we can simulate the emergence of traffic jams by merely assuming differences in the drivers' speed.

Mexican Waves and Standing Ovations

In a Mexican wave, people stand up and sit down from their seats in succession, creating a human wave. In a standing ovation, people stand up to applaud the performance at the end of an excellent play. Both the wave and standing ovation are examples of emergence in stadiums and theaters. No single individual in the audience can create a Mexican wave or a standing ovation. So, how do they emerge? When a few people start, others may follow them. How many need to begin to trigger a wave, and does it matter where they are sitting?

Flocking Behavior

A flock of birds or a school of fish seems to move in unity (as in Figure 1), but no single fish or bird controls or directs the group. What individual behaviors can lead to swarms? See also a later chapter in the book for a model of flocking boids.

Complex adaptive systems can be used to study these emergent phenomena. Complex adaptive systems refer to a group of (locally) interacting agents, who continuously act and react to other agents' actions. The coherent emergent behavior that might occur in a system arises from the local interactions of the agents. In the case of a school of fish, we can explain the swarm behavior as a result of a few simple rules, such as avoiding local crowding, steering

toward the average heading of local fishes, and steering towards the average position of local fishes.

Other examples of complex adaptive systems are:

- **Stock markets**: many traders make decisions on the information known to them and their individual expectations about the market's future movements. They may start selling when they see prices going down (because other traders are selling). Such herding behavior can lead to high volatility on stock markets.
- **Immune systems**: immune systems consist of various mechanisms, including a large population of lymphocytes that detect and destroy pathogens and other intruders in the body. The immune system need to be able to detect new pathogens for the host to survive and therefore need to be able to adapt.
- **Pandemics**: the actions of individuals, wearing masks, washing hands, physically distancing, impact the spread of the virus in a population. No individual can control the spread, and as the COVID-19 pandemic illustrates, severe measures to control the virus have different levels of success.
- **Brains**: The neural system in the brain consists of many neurons that are exchanging information. The interactions of many neurons make it possible for me to write this sentence and ponder about the meaning of life.
- **Ecosystems**: Ecosystems consist of many species that interact by eating other species, distributing nutrients, and pollinating plants. Ecosystems can be seen as complex food webs that can cope with changes in the number of certain species and adapt to changes in climate to a certain extent.
- **Human societies**: When you buy a new iPhone that is

> manufactured in China, with materials derived from African soils, and with software developed by programmers from India, you need to realize that those actions are made by autonomous organizations, firms, and individuals. These many individual actions are guided by rules and agreements we have developed, but there is no simple set of mechanisms that explain these interactions.

Complex adaptive systems and emergence are determined by what we call bottom-up processes. The opposite would be a system of top-down controlled systems where all individual components obey a central commander's rules or a blueprint. Examples are armies and bureaucratic organizations, which are trained to execute orders from higher echelons. They have precise instructions on what to do to meet the goals of the organization as a whole. Taking individual initiatives are not appreciated since it may disrupt the well-trained order in the system.

Bali Irrigation

An interesting example of emergence and complex adaptive systems in social-ecological systems is irrigation in Bali, Indonesia (Figure 2). Farmers who irrigate their fields in Bali have to solve a complex coordination problem that emerges at the intersection of two separate issues: pest outbreaks and water shortage. On the one hand, control of pests is most effective when all rice fields in a watershed have the same schedule of planting rice. The reason is that rice is harvested at the same time, and no food for the pests exists in a larger area to survive. If the harvest of rice was in small fields only, the pest could survive after harvest by moving to a neighboring field where rice is still on the field. On the other hand, the terraces are hydrologically interdependent, with long and fragile systems of tunnels, canals, and aqueducts. Therefore, to avoid water shortage, the irrigators should not plant rice all at the same time.

Figure 2. Rice fields on Bali

To balance the need for coordinated fallow periods and use of water, an elaborate ritual system has been developed that details what rituals and actions should be done on each specific date in each organized group of farmers—called a subak. These actions are related to offerings at temples, ranging from the little temples at the rice terrace level to the temples at the regional level and all the way up to the temple of the high priest Jero Gde, the human representative of the Goddess of the Temple of the Crater Lake. Crater Lake feeds the groundwater system, which is the main water source for irrigating in the entire watershed. These offerings were collected as a counter gift for the use of water that belonged to the Gods.

No person has control over the whole irrigation system. At the temple level, subak leaders come together regularly to exchange information on their irrigation experiences in their subaks. When a particular subak makes a decision that has unfavorable effects on neighboring subaks, leaders of the local temple community come

together to discuss these matters. This may lead to various actions, including the threat of cutting off the water supply to the disrupting subak. When seasonal rainfall is different than expected, subaks leaders may come together to discuss alternative cropping patterns to avoid unfavorable circumstances.

The water temples' function and power were invisible to the planners involved in promoting theGreen Revolution during the 1960s. They regarded agriculture as a purely technical process. Farmers were forced to switch to the miracle rice varieties, which were predicted to lead to three harvests a year, instead of the two harvests of traditional varieties. Farmers were motivated by governmental programs that subsidized the use of fertilizers and pesticides. After the governmental incentive program was started, the farmers continued performing their water temple rituals, but they no longer coincided with the timing of rice-farming activities. Soon after the introduction of the miracle rice, a plague of plant-hoppers caused huge damage to the rice crop. A new variety was introduced, but then a new pest plague hit the farmers. Furthermore, there were problems with water shortage.

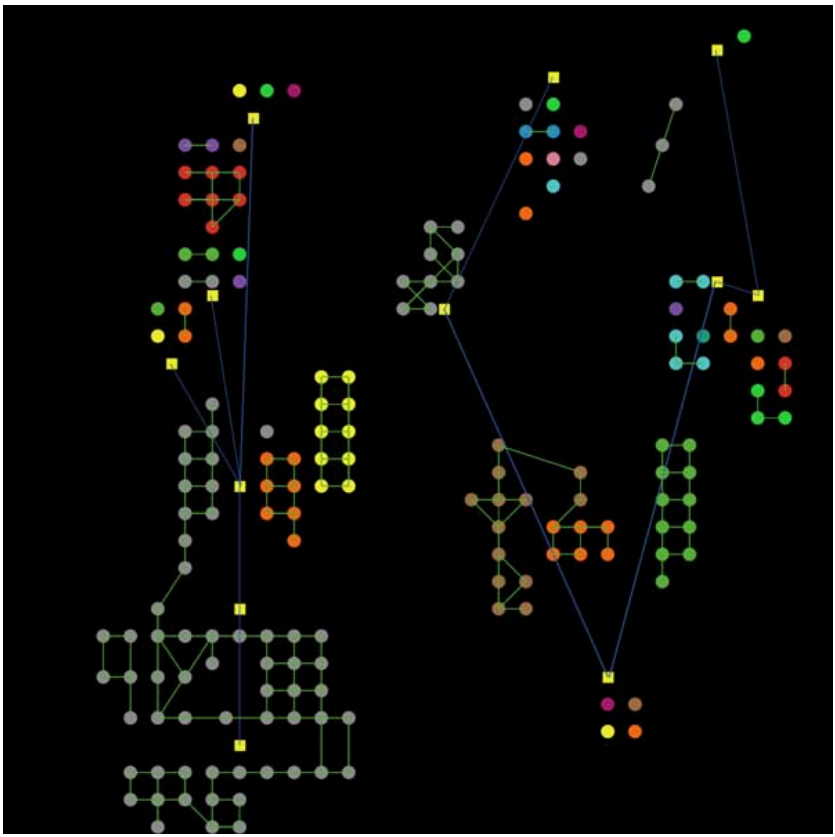

Figure 3. Irrigation network of subaks and temples (Replication of Lansing and Kremer, 1994 available at COMSES Model Library: https://www.comses.net/codebases/2221/releases/1.2.0/).

During the 1980s, an increasing number of farmers wanted to switch back to the old system, but the engineers interpreted this as religious conservatism and resistance to change. It was anthropologist Steve Lansing who unraveled the function of the water temples and was able to convince the financiers of the Green Revolution project on Bali that the irrigation was best coordinated at the level of the subaks with their water temples. Lansing built an agent-based model of the interactions of subak management

strategies and the ecosystem, and the local adaptation of subaks to strategies of neighboring subaks, and showed that for different levels of coordination, from farmer level up to central control, the temple level was the level where decisions could be made to maximize the production of rice (Figure 3). He also showed how the coordination might have been evolved as a result of local interactions. In his agent-based model, agents, representing subaks, make each a decision at the end of the year, which cropping pattern to use for the next year. They will look at their neighboring subaks to determine whether one of them is doing better than themselves. If this is the case, the subak copies the strategy of the better performing subak. Within a few years, the average simulated yield increases to a level that is close to the maximum yield. The result is that there are pockets of subaks with the same cropping pattern, solving the pest problem. These pockets of subaks are small enough to avoid severe water shortages. Steve Lansing, therefore, shows that a complex irrigation network can organize themselves by pure local interactions.

Agent-based Models and Complex Adaptive Systems

Agent-based models and complex adaptive systems are intertwined. When we use agent-based models as a modeling approach, we typically study a complex adaptive system. Many scholars who use agent-based modeling are also influenced by complexity theory and complex systems concepts that cover broader study areas. There are a variety of methodological approaches to studying complex systems besides agent-based systems, such as difference and differential equations, and complex systems also include non-living systems such as pendulums and weather systems.

In all complex systems, we are interested in simple rules that can describe the system's complexity. With complex adaptive systems, we include agents that could evolve, learn, and adapt to changing circumstances. This could be due to genetic evolution, reinforcement of neural pathways, or cultural adaptation.

Moreover, the system's identity may change over time as the agents evolve and adapt to changing circumstances.

Not all complex adaptive systems experience emergent phenomena, and emergence is not the only topic of interest when we use agent-based models. Many macro-level patterns of the studies we do are not well-described emergent phenomena. A land-use pattern of a mosaic of agricultural land and forest could be the consequence of many decades of decisions of households responding to the economic, environmental, and cultural conditions of the system. Such an outcome could be path-dependent and specific to the events that happened in the landscape during that time. As such, the mosaic observed might not be a broader emergent pattern we like to explain by some simple rules.

With a complex adaptive systems perspective, we also implicitly assume that we cannot make precise predictions about the future. We know that small perturbations could have big consequences and that we do not have full knowledge of all the rules and information within the system. Instead of specific predictions, we are typically interested in understanding the mechanisms that explain certain types of system behavior. This will improve our mental models, and can help us to make decisions in complex adaptive systems in everyday life such as stock markets, organizations, social media, etc. We might not be surprised by sudden shifts in dynamics of the system, and changes in behavior. Instead of extrapolating the past to predict the future, we like to explore possible consequences of alternative behavioral rules of the agents and explore alternative futures.

Summary

Complex adaptive systems are systems that are defined by a group of (locally) interacting agents, who continuously act and react to the actions of other agents. Examples of such systems are ant-colonies, immune systems, ecosystems, financial markets, etc.

One of the features of complex adaptive systems is the

occurrence of emergence, which is a macro-level phenomenon resulting from local-level interactions. Examples are ant-trails, immune system response to a virus, flocking behavior of birds, and bubbles in financial markets.

One of the mechanisms in complex adaptive systems that can cause emergent phenomena is stigmergy. Individuals change the environment, which, in turn, affects the behavior of other individuals.

In this book, we will use agent-based models to study complex adaptive systems. We will also use computer simulation to determine how simple rules of local interactions of agents lead to emergent phenomena.

References and Suggested Readings

Gordon, Deborah (1999) Ants at work: How an insect society is organized. New York/London: W.W. Norton & Company.

Holland, John H. (1998) Emergence from chaos to order. New York: Oxford University Press.

Lansing, J. Steve (2006) Perfect order: recognizing complexity in Bali. Princeton, NJ: Princeton University Press.

Lansing, J.Steve and John N. Kremer (1994) Emergent properties of Balinese water temples. American Anthropologist 95 (1): 97-114.

Miller, John H. and Scott Page (2007) Complex Adaptive Systems: An Introduction to Computational Models of Social Life, Princeton University Press.

Mitchell, Melanie (2009) Complexity: A guided tour. New York: Oxford University Press.

Waldrop, MichaelM. (1992) Complexity: The emerging science at the edge of order and chaos. New York: Simon & Schuster.

CHAPTER 2

Introduction to Modeling

Key Concepts

In this chapter, you will

- Get an introduction to models and modeling
- Learn that models are simplifications of reality
- Notice that there are different ways to simplify reality in a meaningful way
- See that you have to make assumptions, but that you will be able to test the effects of the assumptions you make.

In this chapter, many concepts are discussed that might not be familiar to you. We will have some philosophical discussion on how we know what we know and how there are different ways of knowing. By the end of the chapter, you may not have internalized all the concepts, but in the chapters to come, you will recognize the concepts we discuss. In fact, one of the main principles of learning to model is to do modeling, to struggle with the process, and thereby learn how to do it better. Although modeling is a scientific method, to do modeling is sometimes an art.

Epistemology: From ideas to a Conceptual Model

Epistemology is a branch of philosophy that deals with knowledge and the methods of obtaining knowledge. In this chapter, we consider the different ways that scholars reason about their topics of interest as far as this is relevant for agent-based modeling. Since agent-based modeling, like many other modeling approaches, combines knowledge, insights, and tools from various disciplines, it is important to understand that disciplines differ in how they derive knowledge.

Models

Since our focus is on models, let us first discuss what we mean by a model. In fact, there are many kinds of models. A general classification is a distinction between formal mathematical models, formal physical models such as lab rats and fruit flies, and non-formal models such as stories and cartoons. Sometimes people serve as role models for how one might live, such as Nelson Mandela or Claudia Schiffer. In this book, we will only deal with formal mathematical models meaning that they are explicit. Many of them are computer models that can be used to do repeated experiments.

Models are simplifications of reality. To make models, we have to decide what to put in and leave out of the model. What are the essential characteristics of the problem we like to describe? In making the simplifications, we make assumptions about what to simplify. A helpful starting point is to rely on theory in defining your assumptions and use theory to determine what is needed in the model. But there are different theories and different approaches for making assumptions. We can develop formal models by using different approaches. In a broader context, each modeling approach involves its own set of theories, concepts, mathematical techniques, and conventional procedures for constructing and testing models. We can distinguish between deterministic and stochastic models, simulation and optimization models, reductionistic and integrated models, linear and non-

linear models, one agent and multi-agent models, and so-on.

Instead of discussing possible approaches, we want to discuss the difference between the reductionistic Newtonian approach and the complex adaptive systems approach. The fundamental difference is that a Newtonian approach assumes that a system, like the solar system, can be described as a clock. That is, it is mechanical, and we can fully predict what will happen if we just have the right model. Complex adaptive models assume that there is no predictability of the full trajectory of the system. The system does not work like a clock, but like a flock of birds, constantly adjusting to changing context.

Mathematical modeling has long been influenced by physics, which has developed a mechanistic, reversible, reductionistic, and equilibrium-based explanation of the world. This proved to be very successful in calculating trajectories of moving objects (e.g., cannonballs) and predicting the positions of celestial bodies. The work of Isaac Newton, culminating in the Principia Mathematica Philosophiae Naturalis in the late 17th century, was and still is very influential. In fact, Isaac Newton described the world as a machine for which we can predict its development if we understand all the mechanisms. Newton shows that he could explain and predict the movement of planets around the sun correctly. At least correct to an extent given the measurements they had at that moment in time.

The associated rational and mathematical way of describing the world around us was also applied during the last 100 years in the social sciences, economics, and biology. Even though later developments in the natural sciences seriously constrained the applicability of the mechanistic paradigm, its relative simplicity had a great appeal to scientists from various disciplines working with models. However, despite the widespread use of this approach, the mechanistic paradigm is increasingly criticized. The foundations of the mechanistic view – reversibility, reductionism, and equilibrium-

based and controllable experiments – have faded away in the light of a number of "new" scientific insights. Basically, we have to consider whether any of the systems in the life and social sciences can be described as a predictable machine.

First, the discovery of the Second Law of Thermodynamics brought down the notion of reversibility. The Second Law states that the entropy of a closed system is increasing. This means that heat flows from hot to cold so that less useful energy remains. One of the consequences of the Second Law is the irreversibility of system behavior and the single direction of time. History matters!! Changes within systems cannot reverse back just like that (irreversible). This is in contrast to many mechanistic models, in which time can easily be reversed to calculate previous conditions.

If you let a glass fall, it may break. It costs more energy to repair the glass than it requires to break it. As people become older, parts of the body will degenerate. Our body can heal, but scars of time will remain. Except in the case of the main character in the fantasy movie The Curious Case of Benjamin Button where a baby was born looking old, bold and wrinkled, and became more youthful over time.

Second, the equilibrium view of species was brought down by Charles Darwin's book On the Origin of Species during the mid-19th century. The static concept of unchanging species was replaced by a dynamic concept of evolution through natural selection and adaptation, thereby fundamentally changing our view of nature. Natural systems are in continuous disequilibrium, being interdependent, and constantly adapting to changing circumstances.

Third, the theories of quantum mechanics have confronted us with a fundamental uncertainty regarding knowledge about systems, especially on the level of atoms and particles. The uncertainty principle of Werner Heisenberg is well known, stating that it is impossible to simultaneously measure the position in space and momentum (mass times velocity) of any particle. The

statement by Pierre-Simon Laplace in the early 19th century that if every position of every atom was known, the future might be predicted exactly, became, therefore, an illusion. Moreover, the notion of fundamental uncertainty implied that fully controlled experiments are strictly speaking not possible.

Although these developments in the natural sciences changed our perception of the world, (mathematical) models are still frequently based on a mechanistic view of systems. Since the 19th century, scholars have studied systems that are irreversible, stochastic, out-of-equilibrium, and holistic. Yet, since the 1960s, many examples have been found in nature, such as pendulum movement, chemical reactions, weather dynamics, population ecology, and stock markets, which could be better explained by the new tools from complex systems. Complex systems research focuses on finding simple underlying rules of complicated dynamics. Complex adaptive system research is a subsection of complex system research that focuses on systems of various interacting components that lead to emergent phenomena. Instead of using elegant analytical mathematics, as in the Newtonian approach to research, complex adaptive systems research is more dependent on the use of algorithms and computer simulation. When Newton wrote his Principia Mathematica, he used geometry and algebra as the established mathematics to explain his theories, while inventing calculus to better express the mathematics of dynamical systems. Currently, we experience a transition from the common use of differential equations to algorithms since algorithms can better describe many complex adaptive systems.

With the rise of generative Artificial Intelligence and Large Language Models we get a new player in town. We can make generative agent-based models with Large Language Models, and we will discuss this in a later chapter. However, Large Language Models are black boxes for the user. If we want to explicitly express our understanding of systems, build on theory and/or observations, we want to express them in logical and algorithmic

statements. Using Large Language Models to model is largely studying the underlying LLMs itself. The outcomes vary too much between new LLM releases and prompt variations, to be a reliable partner in the modeling process. The coming years will tell whether LLMs become a relevant part of the kind of modeling we pursue in this book as described above.

Deductive and Inductive Reasoning

An important topic in understanding different ways we derive knowledge is the difference between inductive reasoning versus deductive reasoning. With inductive reasoning, we refer to a process where the starting point is the observation of a phenomenon. Specific patterns and regularities might be found, leading to some tentative hypotheses. In the end, these hypotheses may become the basis of more generalized theories. Such an approach is called a 'bottom-up' approach. For example, an ethnographer observes a tribe in the Amazon for several years and finds specific patterns. Later she may compare her findings with other hunter-gather communities, which may lead to a more generalizable theory.

Observation -> pattern -> tentative hypothesis -> theory

Many social and life scientists use the inductive approach. It focuses on observation in situations where one does not know what to look for. A trained social and life scientist can observe interesting and surprising behaviors that need explanation.

The deductive reasoning approach follows the opposite direction. One starts with a theory about the topic of interest and defines hypotheses in order to test the theory. Controlled experiments may be defined and executed to derive observations that one can compare with the predicted behavior. This may lead to confirmation or falsification of the original theory. An example of deductive reasoning is seen with an experimental social scientist who tests the prediction that humans will take into account others' welfare in making decisions. An experiment is developed to test the theory in a controlled setting where the subjects have to make

decisions. The results may confirm or falsify the original theory, and one can compare which theories best explain the observations.

Theory -> hypothesis -> observation -> confirmation

Both approaches are equally valid methods deriving knowledge, although some disciplines are focused on induction (anthropology) while others are focused more on deduction (economics). It is very reasonable to assume one needs to use both approaches to understand complex social processes. In fact, modeling is seen as a third approach to derive knowledge by combining both approaches.

From a modeling perspective, we can observe the two different approaches. Some scholars develop models with a particular data set taken from a case study as their starting point. One first derives statistics from the empirical case and then develops a model that may fit the data. When they have developed a model that fits the observations, this model may serve as the first step toward a theory that one can test in future case studies. The Bali irrigation case of chapter 1 is an example of this approach.

The other approach starts with a general abstract model using rules based on existing rules. The question being asked with this kind of approach is what kind of patterns will evolve when the model is run. Sometimes we derive new insights by testing existing theories or adjustments of existing theories in new contexts as we can do with simulation models. For example, Thomas Schelling showed with a model of pennies and dimes that segregation of groups was possible even if agents were tolerant of living among agents of the other type. We will discuss Schelling's model in detail in a later chapter.

Both inductive and deductive reasoning assume that we want to develop and test a theory. In the era of big data and machine learning algorithms can predict your shopping behavior and learn how to play chess. Some scholars start making an argument for the end of theory since there is no need for understanding as machine learning outperforms theoretical approaches. This might

be true for repeated activities like shopping behavior for which a high volume of good quality data can be collected, but this will not help to derive projections on how to transition to a low carbon economy, manage the COVID-19 crisis or define policies to reduce religious terrorism. For those, we need understanding, and therefore theory, not big data, will be a guiding principle for agent-based modeling.

Dynamics and Scale

The two hardest concepts in social science, maybe in science in general, are dynamics and scale. In fact, we don't even have very clear descriptions of these concepts. If we look at a car, we can measure the speed of the car. In the world of Newtonian mechanics, objects follow the laws Newton uncovered, and we can measure speed and acceleration with high accuracy. But in social systems, change happens in people's minds. The change of beliefs, expectations, understandings, or preferences all affects our decision making. Such changes can only be measured indirectly, for example, via experiments and surveys. It does not help that our subjects of interest, in contrast to Newton's atoms, reflect on their actions. If we ask the same person to fill in the same survey one week apart, we likely get differences in responses, even if the respondent is answering the survey truthfully.

We can measure a census of citizens, sale statistics, email patterns, traffic flows, etc. as indirect information about decisions people make. Using statistical models, we can make predictions if people continue the behavior they did in the past. Statistical models cannot well-capture what happens if the behavior is changing. For example, will people react to information about a possible new pandemic in the same way they reacted to the news of the most recent flu epidemic? In many models in social simulation, we describe the decision process itself. This makes it possible to capture a broader diversity of behavior. We would be able to predict the future better if people did not change their behavior. Statistical models are superior for doing just that. But

humans can anticipate possible behavior and change their behavior, making statistical models obsolete during periods of change. For complexity scholars studying financial markets, the 2008 financial crisis was not a surprise since they had observed that type of behavior – herding of decision making leading to bubbles in markets – regularly in their simulation models.

Another question to consider when studying social systems is at what level we describe the entities of the system? One can look at individuals, couples, families, firms, organizations, communities, nations, etc. Although these seem clear categories, in practice, boundaries can be fluid. For all these levels, we can look at entities that interact with each other. Of course, actual systems consist of entities at different scales of organization. To make it more complicated, a person might be a member of a family, a firm, an organization, a community, and a nation, and have different roles within those entities. Suppose we have two persons A and B, person A might be the chairperson of a sports organization in which person B participates, but person B is person A's boss in the firm.

Studying social systems is complicated since it is difficult to follow a system's changing behavior at different levels of scale. When we develop models, we will sometimes consider different scales, such as individuals and firms, but it will be a challenge to make models that include agents of different levels of scale.

The Modeling Process

How do you create a formal model? You don't start by sitting at a computer writing a program. Rather, it is useful to follow a set of actions that provides you with a more reasoned approach to the model design.

First, you need to define the scope of your research. What is the question you want to address? You may brainstorm in a group or by yourself what the future model needs to be able to do? Is it used to test a specific hypothesis, or is it used as a planning tool in collaboration with stakeholders? Do you want the model to be

interactive with visual output, or to run the model often for many different assumptions?

Suppose you are interested in studying when a movie becomes a hit. The question then becomes how to make a model that can predict whether a movie will become a hit based on the characteristics of that movie. Such a model can be developed from available statistics, and a statistical model seems to be an appropriate approach. However, if one wants to understand the "ecology" of movies, why one movie attracts lots of consumers while a similar movie does not, an agent-based model might be more suitable.

One of the interesting statistics about movies is that most movies derive the highest earnings in their first week. After a few weeks, the movie stops attracting viewers. If people go to a movie because they read good reviews of the movie, you would expect that with successful movies, there would be an increase in visitors over the weeks. However, this is a rare pattern in the data. In Figure 1, we depict the U.S. returns of two movies that were released in the middle of November 2008. The James Bond movie Quantum of Solace earned half of the total earnings in the first week after its release (82 million dollars). The returns fell rapidly after the first week. This is in contrast to the Oscar-winning movie Slumdog Millionaire which started with very modest earnings. It gained momentum during the Christmas season and got an extra boost when it was nominated for 10 Oscars. The highest revenues were derived the week after the movie won the Oscar for the best motion picture.

The pattern of Slumdog Millionaire is sporadic in movie ecology. Most movies follow the pattern of Quantum of Solace. Most people want to see a movie when it first becomes available. The same phenomenon, highest sales at the moment of release, is also found in other information products like books (consider the sales of the Harry Potter books).

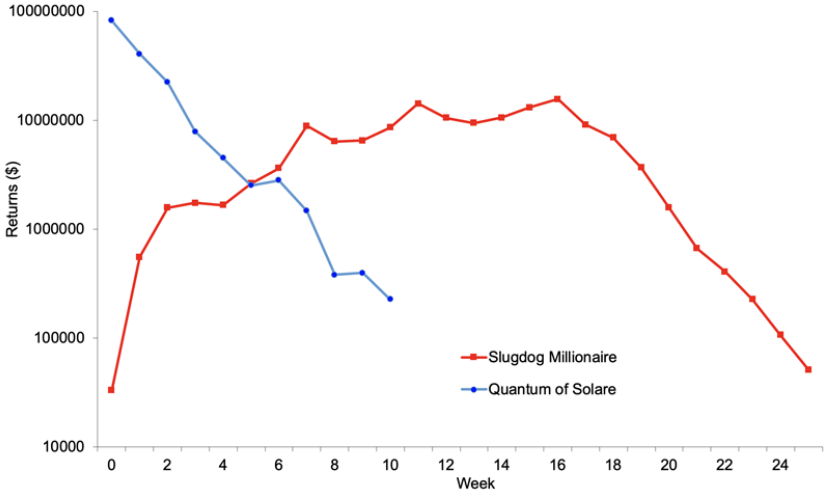

Figure 1. The returns in dollars on the U.S. market for each week in the cinema.

When you get some ideas of what you want to work through, you can start to define the components of the model. It is important to "keep it simple, stupid." So, try to identify only the most critical components and processes. What are the agents? What are their attributes and their interaction with other agents and their environment?

For the movie example, one may consider at least individuals as potential moviegoers, movies, and producers of movies as agents. Individuals have a variety of preferences, limited time and resources, friends and family with whom they like to go to the movies, a distance from nearby cinemas, etc. Movies have different characteristics such as genre, actors, and a budget for movie making and advertisements. Producers of movies need to decide which kind of movies to produce and when to release them to derive a profit. You can add more and more details, such as cinemas, actors, media, and advertisements. The challenge is to find the right balance between creating a simple model and having enough content to address the questions you have. This will be an

iterative process. You may think you have a simple model, but find out that your model results are difficult to understand, and you need to rethink how to simplify it even more.

After these steps, it is time to look at the literature. Have other people studied similar questions to the one you want to address? You can use Google Scholar, "Web of Knowledge," or the Catalog App to look for specific keywords to find relevant models. If you find several useful papers, find out the pros and cons of these other papers. What can you learn from these papers? What would you like to do differently? Now you can start to define more precisely the hypotheses you want to explore.

There is a substantial literature on movie attendance modeling across different disciplines, such as economics (De Vany, 2003), and marketing (Elberse and Eliashberg, 2003). There are also several agent-based models developed already, such as Delre et al. (2008).

Note that we use a deductive approach in our model design. An inductive approach is possible for social simulation in the following sense. You formulate a number of agents and their interactions and let the model run, observing what happens. Sometimes an explorative approach is used, for example, by changing existing models and adding new processes to explore the consequences. Since the model process that accords the deductive approach is stricter, we will use this approach to discuss the modeling process.

Now you can become more specific on model components. You may start developing a flow chart of the conceptual model, by which we mean a set of boxes representing variables, and arrows representing relationships. What are the specific attributes of the agents, the cells, and other components of the model? What kind of learning model do you like to use? Do the agents derive information without observation error? Etc.

For the movie market model, we need to start thinking about the interactions between individuals, movies, and producers (Figure 2). Potential moviegoers can be influenced by advertisements, the attributes of the movie, and by the opinions and behaviors of other

people in their social network. The producer needs to decide what kind of movie to make, how much to spend on advertisements and when to release it. If the producer has a good portfolio, will it survive the competition? Do producers specialize in certain movies or maintain an extensive portfolio? What can explain the fact that after the first week of its release, the movie will have attracted the majority of its audience?

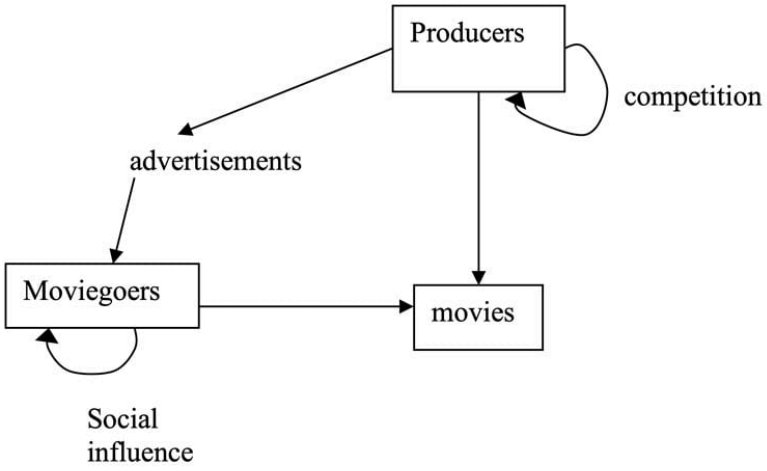

Figure 2. Flow diagram of a possible model of movies.

Developing flow diagrams leads to a detailed plan of action, and you can finally start the coding of the model. What platform do you want to use? The choice for a particular platform depends on you and your collaborators' familiarity with the platform, the technical requirements of your model (visualization, connection with Geographic Information Systems), and the audience (students, stakeholders, and other scientists). When you have decided on the platform, it is useful to see if there are models available that use similar processes as your model. You can use such a model as your starting point. When you have implemented the model, it is vital to test the code (verification). For example, you simulate situations

where you know the result and confirm that your model produces the correct results.

When you have coded the model, it is important to write down the model specification on paper, including all the assumptions, relevant literature, and parameter values you are using. This will be important when you start writing an article about your model analysis later.

The main use of a computational model is the analysis of its behavior for different assumptions. In a mathematical model, we can frequently describe the system's behavior in closed form, which is an equilibrium described by a set of equations. This means that just a few equations describe all the possible situations of the system. This is not possible with simulation models. We can implement the behavior of a number of agents with just a few rules, but to understand what the consequences are, we have to do simulations for quite a number of different parameter settings. We need to do a sufficient number of simulations to get a good grasp of the system's behavior. We will discuss later how to use statistical tools to describe how many simulations and ways there are to explore the parameter space.

Summary

Models are simplifications of real systems, and to simplify, we have to make assumptions. There are different ways to make these assumptions, and such decisions are made based on the training and discipline of the modeler. Models should not be seen as holy grails that predict the future but as tools to explore different assumptions systematically.

References and Suggested Readings

Delre, Sebastiano A., Thijs L.J. Broekhuizen, and Wander Jager (2008). The effect of social influence on market inequalities in the motion picture industry. Advances in Complex Systems 11(2): 273-287.

De Vany, Arthur (2003). Hollywood economics: How extreme uncertainty shapes the film industry. London; New York: Routledge.

Elberse, Anita and Jehoshua Eliashberg (2003). Demand and supply dynamics for sequentially released products in international markets: The case of motion pictures. Marketing Science 22(3): 329-354.

Epstein, Joshua M. (2008). 'Why Model?'. *Journal of Artificial Societies and Social Simulation* 11(4)12.

Miller, John and Scott E. Page (2007) Complex Adaptive Systems: An Introduction to Computational Models of Social Life, Princeton University Press.

CHAPTER 3

The Logic of Programming

Key Concepts

In this chapter, you will

1. be introduced to functions, programs, and algorithms,
2. see common components of algorithms,
3. learn that random numbers are not random in a computer,
4. see that the order in which you give commands to a computer matter.

A computer is basically a large number of electric switches that are either turned on or off. To program a computer, we must develop a way to communicate in binary code: 0 and 1. Computer languages like Java, C++, or Python do this kind of translation for you. Nevertheless, it is helpful to know that a computer is just a large number of on/off statements. To get the computer to do what you want, you have to be precise in the commands you give.

This chapter provides a background to do programming before focusing on NetLogo, the programming platform used in this book, in the next chapter.

Functions

Suppose we want to model the number of students at the university to investigate how many new classrooms need to be built in the coming years. We will need to predict how the number of students will change over time. Therefore, our main focus will be the number of students, which is represented as *NoS.* The number of students in 2000 is defined as *NoS(2000)*, In 2001, some students graduated, others dropped out, and new students came to the university. To describe the number of students in 2001, we may have something like:

NoS(2001) = NoS(2000) – Graduated – Droppedout + Transfers + Newstudents

This relationship will be the same for each year. As such, we can define a function for *NoS(t)*.

In more general terms, most agent-based models describe how values of variables change during the iterations of the simulation. A v*ariable* is an attribute of a system that can change over time. With a *system*, we refer to a set of interacting or interdependent entities, forming an integrated whole. With *iterations*, we mean the execution of a set of instructions that are to be repeated.

Given is a variable of interest *x,* which might refer to the population size, level of financial capital, or the level of natural capital. Such a variable is called a *state variable*, and a dynamic model describes how the value of *x* changes over time. In the simulation models discussed in this book, time is considered to be discrete. Hence, given a time step t for which *x(t)* is defined, we need to redefine *x(t+1)*. To determine the relationship between the current and the future state of the system, we formulate a difference equation:

x(t+1) = f(x(t))

where f*()* is a function or a variable *x*. More specifically, the function *f(t)* may consist of a number of parameters *p*, which determines the shape of the function:

x(t+1) = f(x(t), p).

Examples of parameters are the growth or death rate of a species, interest rate, etc.

Programs

In this book, we will learn how to develop computer programs to simulate social, ecological, and social-ecological phenomena. In this chapter, we introduce some basic concepts of programming. A computer program consists of data and algorithms. Data refers to quantitative information of the program, while an algorithm is a set of ordered steps to solve a problem. In fact, a program is like a recipe. Let's look at a recipe to prepare a Reuben sandwich.

Ingredients

- 0.5 pound (225g) sliced corned beef
- 0.25 cup (60ml) drained sauerkraut
- 2 tablespoons chopped sweet onion
- 2 tablespoons creamy Russian dressing
- 2 slices rye bread
- 1-3 slices Swiss cheese
- chopped parsley
- 1 tablespoon butter

Procedure

1. Combine the sauerkraut, onion, and parsley
2. Spread the dressing on two slices of bread
3. Pile layers of corned beef, cheese, and sauerkraut mixture on one slice
4. Add the second slice of bread
5. Butter the outsider of the bread
6. Grill until lightly browned, roughly 5 minutes

The ingredients are like the input data for a program, while the procedure is the algorithm to make a Reuben sandwich given the input data. Additional data are used in the algorithm, such as 5 minutes in step 6 of the procedure, which is like a parameter value in a program. Although a recipe is like a program, different persons may generate different variations of the sandwich. With a computer program, the instructions have to be clear enough so that each computer and operating system, will produce exactly the same results. There is no room for different interpretations. In practice, this can make computer programming a time-consuming and frustrating experience since you will get error messages or wrong results if you don't give the right commands.

The concept of algorithms has been around before computers were developed. In fact, the term algorithm was named after the 9th century by Persian mathematician Muḥammad ibn Mūsā al-Khwārizmī since Algoritmi was the Latinization of Al-Khwarizmi's name. Al-Khwarizmi was instrumental in introducing Europe to the Indian numerical system and algebra.

Example of Algorithms: Sorting

Suppose you have the following list of numbers:

4 2 1 3

You are asked to develop an algorithm that sorts this list of numbers from small to large. This algorithm should also be potentially applied to any other list of numbers. What would such an algorithm be? A simple algorithm would be the so-called bubble-sort. This algorithm repeatedly steps through the list to be sorted, comparing two items at a time and swap them if they are in the wrong order. This process is repeated until a pass through the list does not lead to any more swaps. The list is then sorted.

2 4 1 3 swap
2 **1 4** 3 swap
2 1 **3 4** swap
1 2 3 4 swap

1 **2 3** 4
1 2 **3 4**
1 2 3 4
1 **2 3** 4
1 2 **3 4**

Bubble sort is a simple but very inefficient algorithm since it takes many repeats to go through the list to get sorted. The duration of the algorithm grows exponentially with the length of the list. Another sorting algorithm invented by John von Neumann, which is much more efficient, is merge sort. It is based on the observation that a shortlist can be sorted quickly, and one may split up the list into smaller units, and then merge the smaller lists into a bigger list.

4 2 1 3
2 4 1 3 swap
2 4 **1 3**

Now it compares the first of both lists

2 4 **1** 3 : 1 2
2 4 1 **3**: 1 2 3
2 **4** 1 **3**: 1 2 3 4

The computer had to do 9 comparisons with the bubble sort and only 5 with the Neumann algorithm. For larger lists, the efficiency of Neumann's algorithm will become more prominent.

Components of Algorithms

We will now discuss several concepts that are used in developing algorithms.

IF THEN ELSE

In everyday situations, you have many different choices: if the traffic light is green, you cross the street; otherwise, you wait. If you want to send an e-mail, you need to include the correct e-mail address and be connected to the internet; otherwise, it will not reach the destination.

In programs, we have many of these situations too, which are stated more formally as IF a condition is true THEN do this ELSE do that. This procedure is used to let algorithms do tasks that are

dependent on the state of the system. For example, an agent will move randomly except when the agent sees food on the cell in front of itself. In this case, the agent will move forward. In this example, the situation is stated.

IF food on next cell THEN move forward ELSE move randomly

Another example shows how you can avoid dividing by zero. Given are the variable x, and parameters a and b.

IF a != 0 THEN x = b / a ELSE write "division by zero", where != means "not equal to"

For example, in bubble sort, you can have the following use of IF THEN ELSE.

```
IF a > b THEN

   c = b

   b = a

   a = c

ELSE

   a = a

where a and b are two items in the list that are swapped if
```

Boolean

This is a variable that is defined to be FALSE or TRUE.

A boolean variable has only two possible values, TRUE or FALSE. This means that if a boolean variable x is not TRUE, it will be FALSE.

Suppose *x* is FALSE. What is the value of *a* in the following statement?

```
IF x THEN a = 2 ELSE a = 1
```

Since *x* is FALSE, *a* is equal to 1.

While loop

Sometimes you want to repeat a set of commands until a

particular condition is met. For example, you are searching through a list of numbers until you find a specific value. You move to the next item in the list until you find the item you are looking for. You can implement this by using while loops.

Using the search example you can implement this as follows

```
position = 1

WHILE a != searcha

[

   a = list [position]

   position = position + 1
]
```

where searcha is the value looked for. Recall that != means "unequal to". One moves through the list, looking at the value of each item until searcha is found.

What will be the value of x in the following while loop?

```
A = 1

WHILE A < 10

[

   x = A * 2

   A = A + 2

]
```

The answer X will have values 2, 6, 10, 14, 18 and then the while loop will be ended.

Recursion

Recursion implies a procedure that calls itself and it is commonly

used in computer programs. A simple example is to calculate n! by the following procedure factorial.

```
Factorial[N]

   IF N <= 1 THEN fact = 1 ELSE fact = N * Factorial[N-1]
```

Thus if we want to calculate 4!, we will start with Factorial[4].

Factorial[4] leads to fact = 4 * Factorial[3]
Factorial[3] leads to fact = 3 * Factorial[2]
Factorial[2] leads to fact = 2 * Factorial[1]
Factorial[1] leads to fact = 1
Then it jumps back yo finish Factorial[2]: fact=2 * 1 = 2
Then it jumps back to finish Factorial[3]: fact=3 * 2 = 6
Finally it can finish Factorial[4]: fact = 4 * 6 = 24
Thus 4! = 24

Random numbers

Random numbers in simulation models are not truly random. Instead, they are generated by the so-called random number generators. In fact, the functions used to generate random numbers are deterministic and may, therefore, repeat the same sequence after millions of 'random numbers.' An example is the linear congruential generator that generates sequences of numbers x by using the next equation.

```
x(t+1) = (A * x(t) + B) mod M
```

where the A, B and M are pre-selected constants. Note that M represents the upper level of the sequence of random numbers. A and B are defined in such a way that they meet desirable statistical properties. An example which is commonly used is A = 1664525, B = 1013904223, and M = 32. Another number that needs to be specific is the seed, which is the value of x at the start of the sequence.

Suppose you start with x = 1, then the next step we get

```
x = (1664525 * 1 + 1013904223) mod 32
```

Since mod means modulo which finds the remainder of the division of one number by another we get

```
x = 1015568748 mod 32 = 12
```

If we continue we get the sequence 1, 12, 27, 30, 5, 0, 31, 18, 9, 20, 3, ..

Any idea what the next number will be?

When you specify seeds in simulations, you can replicate the same results, even though you use random numbers. Sometimes the seeds are initialized using the computer's real-time clock, which may provide enough randomness for simulation purposes, but the individual results can not be replicated.

The function random() in most programming languages produces a random number between 0 and 1 according to a uniform distribution. When the underlying sequence of random numbers falls between 0 and M, the random number is divided by M to produce a random number between 0 and 1.

The order of commands

The order in which a program will execute the commands can be important. Suppose I have two agents. Each agent has a money account and the program gives one of the agents 1 dollar. What will happen in each of the following two algorithms?

Algorithm 1:

For Agent 1 [Account(t+1) = Account(t) + 1]

For Agent 2 [Account(t+1) = Account(t) + 0]

Algorithm 2:

Flip a coin, IF head THEN [A = agent 1, B = agent 2] ELSE [A = agent 2, B = agent 1]

For A [Account(t+1) = Account(t) + 1]

For B [Account(t+1) = Account(t) + 0]

Object-Oriented Programming

In this book, we use NetLogo, a modeling program designed specifically for agent-based modeling and developed using Java. Java is a so-called object-oriented programming language. NetLogo

provides a user-friendly shell around Java. NetLogo is also an object-oriented programming language. To understand how to read and write programs, we need to discuss briefly what object-oriented programming means.

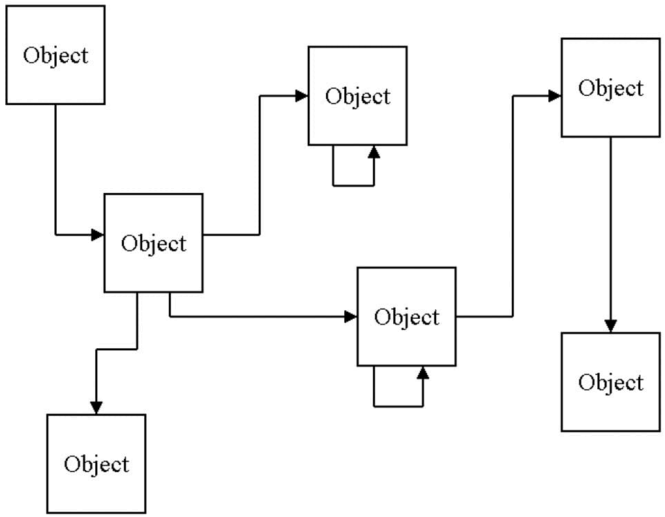

Figure 1. Flow diagram of objects in a computer program.

A program is build up by objects (Figure 1). These objects are little pieces of code that are activated during the simulation. For example, you make a model of ants walking around in an environment, searching for food, and bringing food back to the next. Instead of making one long list of commands, you split it up into components such as: how to walk when searching, how to walk back to the nest, how to recognize food, how to drop pheromone, etc. These components are self-sustained and can be called upon by different places in the program. If the user starts the program, the object in the left top corner will be activated (in NetLogo, the objects are called procedures, and the starting procedure is often called 'go'). This object will call another object, and based on what

happens in the object, one of the different objects is called. Some objects call themselves. This is an example of recursion.

In your NetLogo models, you will see that you first need to activate 'setup', a set of calculations to clean up the memory define the number of agents and cells, and the starting values of the variables. Only when this has happened, the procedure 'go' can be activated, which will run the model through various iterations.

The Art of Programming

Programming can be a time consuming and frustrating process. Especially in the beginning, the program never seems to do what you intended it to do. By practicing regularly, you see that you make common mistakes (typos, forgetting a bracket [], etc.) and learn to debug, you will code better over time. Nowadays we can also make use of generative AI as an assistant to help you generate code, which still need to be evaluated and tested whether it works properly.

If you want to make a new program, it will help to follow some best practices. It is important not to start programming immediately. First, define the program and outline it in words or a drawing. Once you have identified what components need to be included, then you can write your model in pseudo-code, which is a high-level description of your model. Once you have completed these steps, then you can start programming.

If possible, start with some components that you can test before you add another component. By the end of this process, your program should be working as you intend it to. A common understanding among programmers is that there are no programs without errors. Therefore it will be helpful to do many tests, such as running conditions for which you know the answer to see whether your model is accurate. You can also let somebody else work with your model. Another way to check your model is to write clear documentation.

Summary

In this chapter, we introduced some concepts that we will use

during the rest of the book. Computer programs consist of algorithms. Algorithms are a set of commands you give to the computer. Computer programs are very strict since the commands are taken literally. It will take a lot of practice to learn to program. That is, you need to do many exercises to learn to program.

CHAPTER 4

Introduction to NetLogo

Key Concepts

In this chapter, you will

- be introduced to the programming language Netlogo,
- get practical examples to build your first models.

We use the NetLogo package to implement the agent-based models discussed in this book. This package can be downloaded for free from here. We use Netlogo version 7 -(7.0.4 to be precise) in this book. The NetLogo website has a lot of information available for novices like tutorials and example models.

Model Structures

A NetLogo model, like other platforms for agent-based models, consists of algorithms that produce a sequence of commands to calculate changes in agents' attributes. First, a model needs to be initialized, which means that all attributes of agents and environmental variables are defined and have initial values. In NetLogo, people typically use the procedure "setup" to do this. When you look at one of the many demo models in the NetLogo library, you find the button "setup" in most interfaces. Clicking on that button initializes all the relevant variables of the model.

When a model is initialized, the model continues by calculating the updates of the attributes of agents and the environment. This is typically started by clicking on the button “go” in the interface.

When you click on “go”, you start a sequence of calculations. A model can be described as follows:

```
F(x,p,t)=F(x,p,t-1)
```

where F is a model, x the state variables of the model, p the parameters of the model, and t refers to the time step during the simulation, and F(x,p,0) is the initialized version of the model.

Most models are built upon many different procedures. Procedure f may call procedures g and h. An important aspect of simulation models is the sequence in which procedures are called, and thus the order in which variables of the model are updated. Suppose you have procedures *die*, *eat,* and *metabolism*. Suppose you first call the procedure *die* to check whether the agent has sufficient energy. If not enough energy is available, the agent is removed from the model. Then the procedure *eat* is called, and the remaining agents may derive more energy before the procedure *metabolism* is called. Compare this sequence with the case when first the procedure *eat* is called, then *die,* and finally, *metabolism*. You can tell that the order in which the agents are updated effect which agents remain in the model. Hence, the sequence in which you call the procedures is essential.

The scheduling is not only important for the order in which you call procedures, but also the order in which you update the agents. Suppose you always update agents in the same order, and they have to look for resources. Then the first agent has an advantage in finding available food, while the last agent may always experience a scarcity of resources. Such a fixed order leads to artifacts in the model. Therefore you see that the order in which agents are updated is typically random. When in NetLogo, you call ask foragers [], forager agents are updated randomly.

First Steps with NetLogo

How can you learn to use NetLogo? First, you can follow the tutorial on the NetLogo website. Second, you can explore a large number of existing models in the NetLogo model library. You can explore the behavior of a model by running the model for different assumptions, such as different values of parameters. Let's use the Rabbits Grass Weeds model in the model library (File – Models Library – Biology).

In this model, rabbits move rounds freely on the landscape and consume weeds and grass. The rabbits have a metabolism of say 0.5 energy units every time step. By eating grass, and weeds the rabbits derive new energy. When the accumulated energy of an agent is above the birth-threshold, the agent will generate offspring. The child's generation is implemented as adding a clone of the agent in a nearby cell and providing half the energy of the parent to the cloned child. As a result, the number of rabbits in the population evolves over time, as well as grass and weed, which reappear by a fixed chance on an empty cell.

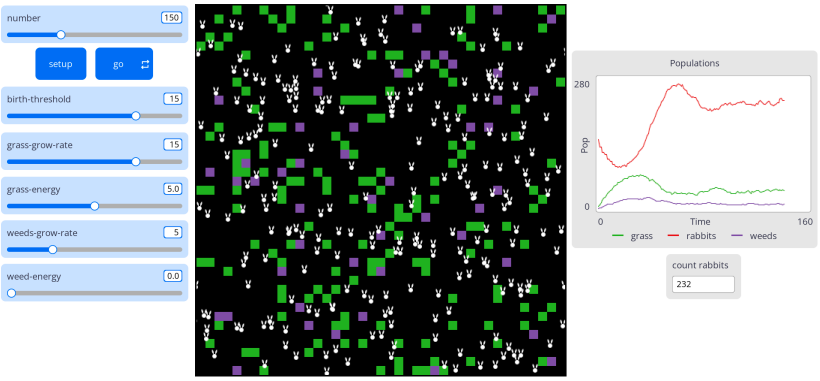

Figure 1. NetLogo interface of the Rabbits Grass Weeds model

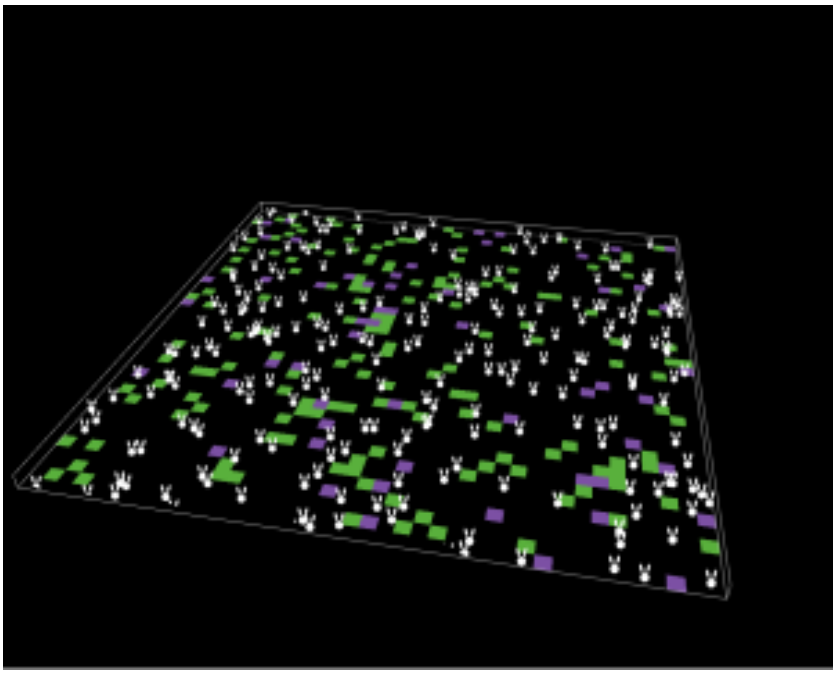

Figure 2. Example of a 3-dimensional view of the Rabbit Grass Weed model.

The basic user interface includes the "interface," which gives the graphical output of the model (as depicted in Figure 1), the "info," which provides the background information of the model, and "Code," which contains the source code of the model. The basic interface, as depicted in Figure 1, shows more components. A few are important to mention. "File" provides the option to save the model, start a new model, or find a model in the model library. "Tools" include several helpful tools like "Shapes editor" to select a shape to represent an agent, "BehaviorSpace" to run a large number of simulations and store the results in a data file, and "3D view" which allows you to visualize the model output in 3D (Figure 2). "Help" includes a useful link to the user manual. In the detailed user manual, there are discussions on a number of sample models, as well as helpful tutorials that go through the basics of

NetLogo. The manual also includes detailed information on how to develop the interface as well as programming guidelines. Finally, the manual consists of a number of tutorials on the use of features like "Shape Editor," "Behavior Space," "Sound," and "System Dynamics."

To use the interface in Figure 1, you click on setup, which will run the command to initialize the model. After this, you can click on "go" to run the model. You can change the parameters during the simulation and see the effects directly.

Programming with NetLogo

Learning to program in NetLogo is mainly a process of learning-by-doing. By working through tutorials and example models, you will become familiar with the basics of the language and where you have to look for solutions to the problems you encounter. Note that you will see NetLogo code using different colors. Code other than black are NetLogo specific commands.

A few basic elements of the NetLogo language are discussed in the following paragraphs. Agents are called turtles in NetLogo. NetLogo has been based on follow up languages of Logo, a procedural programming language developed in the 1960s. The idea in the development of Logo is to have a simple language to steer an agent or robot, the turtle. For example, a turtle with a pen strapped to it can be instructed to do simple things like moving forward 100 spaces or turn around. The figure below shows you can create a square by having the following sequence of commands.

```
Forward 50

Right 90

Forward 50

Right 90

Forward 50
```

```
Right 90

Forward 50

Right 90
```

Where Right 90 means that the turtle will turn right 90 degrees.

From these building blocks, you can build more complex shapes like squares, triangles, circles, etc. A software package like NetLogo simulate populations of turtles and provide them with many more commands. Turtles have coordinates (xcor, ycor), and orientation (heading), and they can move around in the environment of patches.

The cells of the spatial explicit environment are called patches. Patches have coordinates (pxcor, pycor), neighbors, and a color. Patches can also include more information like whether it is empty, has an agent on it, or contains grass or weed (like in Figure 1).

There is also an observer perspective. From this perspective, you can define aggregates of the patches and turtles, and define the order in which certain commands need to be executed.

The program consists of a number of procedures which define the actions of the turtles and patches, and in which order they are executed. A procedure starts with to and ends with end. As an example, we define below two example procedures set and go.

```
to setup
  ca       ; clear the world
  crt 10   ; make 10 new turtles
end



to go
  ask turtles
  [
    fd 1           ; all turtles move forward one patch
    rt random 10   ; .. and turn their head a random amount
```

```
  ]
end
```

We recommend that you implement the model above before you continue

There are three main types of procedures in NetLogo

Commands are a list of actions for the agents to carry out. A command always starts with to and ends with end, hence a typical command looks like:

```
to (commandname)
   (actions)
end
```

In the following example, the name of the command is draw-polygon, and it uses values of global variables num-sides and size. The other actions (pd, repeat, fd, rt) are primitives, as discussed below. Look up the meaning of these primitives to understand what the command draw-polygon is will be doing.

```
to draw-polygon
  pd
  repeat num-sides
  [
    fd size
    rt (360 / num-sides)
  ]
end
```

Reporters are some specific commands. They report the value of a function with a specific input. Sometimes you have a set of actions that you use in different parts of the model. In such a case, it is useful to define this sequence as a function. The general layout for a reporter is

```
to-report (namereporter) [ inputvariables]
  (actions)
end
```

See for example a function that defines the absolute value of a number,

```
to-report absolute-value [number]
  ifelse number >= 0
  [report number]
  [report 0 - number]
end
```

Primitives are built-in reporters and commands. The developers of NetLogo have defined a number of common reports and commands that many people use, and then build those directly into NetLogo. If you look at the primitives directory you see a large set of primitives like turtles, turtles-here, ask, clear-all, etc. Learning which primitives are available and how to use them will take some practice. The best way to learn this is by looking at example models and see how other models have implemented certain problems.

Variables that are globally defined can be used everywhere in the program and are defined at the top of the program.

```
globals [parameter]
```

or are defined as a slider or switch in the interface.

Also, the attributes of turtles and patches are defined globally:

```
turtles-own [attribute]
```

```
patches-own [attribute]
```

To update a variable you use the command set, for example

```
set year year + 1
```

You can also define a local variable within a procedure by

```
let variable 0
```

Note that a local variable can only be used within a procedure.

To update a variable for all turtles use the command ask.

You can update the color for all turtles:

```
ask turtles [set color red]
```

or of all patches

```
ask patches [set pcolor red]
```

You can also change the color of the patches on which there are turtles

```
ask turtles [set pcolor red]
```

or a specific turtle

```
ask turtle 5 [set color green]
```

or

```
set color color-of turtle 5 green
```

or patch

```
ask patch 2 3 [set pcolor green]
```

A more complicated procedure is

```
ask turtles
[
  if (attributes > threshold)
  [
    set attribute attribute - 1
    fd 1
  ]
]
```

In the procedure above, the agent is not doing anything if the attribute is less than the threshold. In the following procedure, we adjust the earlier procedure to include that agents make a different decision if the attribute is smaller or equal to the threshold. We

also will visualize this in the graphical interface by giving the turtle different colors for each condition.

```
ask turtles
[
  ifelse (attribute > threshold)
  [
    set attribute attribute - 1
    set color white
    rt random-float 360
    fd 1
  ][
    set color red
    rt random-float 90
    fd 2
  ]
]
```

Agentsets are a subset of the agents, for example all red turtles:

```
turtles with [color = red]
```

or all the red turtles on the current patch:

```
turtles-here with [color = red]
```

Another example is all the patches at the right part of the screen

```
patches with [pxcor > 0]
```

From the perspective of the 'caller', a turtle or patch, you can define an agentset of all turtles within a certain radius:

```
turtles in-radius 3
```

Defining four neighboring patches (north, south, east and west):

```
patches at-points [[1 0] [0 1] [-1 0] [0 -1]]
```

but this can also be defined easier with a primitive neighbors4.

What are agentsets for? Well you may want to update a specific set of agents, like

```
ask [..]
```

or check whether there is any specific agentset:

```
show any?
```

or count how large the agentset is

```
show count
```

A specific example is the following where the turtle with the lowest fitness is removed

```
ask min-one-of turtles [fitness][die]
```

Special agentsets are breeds. In fact, you define them typically in the setup of your model, like

```
breed [consumers consumer]
```

If you look into the primitive dictionary you see a number of specific primitives for breeds like

```
create-<breed>
create-custom-<breed>
<breed>-here
<breed>-at
```

So that you can use it like

```
ask turtle 100 [if breed = consumer [..]]
```

or, if you want to change the breed of a turtle

```
ask turtle 100 [set breed consumer]
```

As mentioned before, learning to use NetLogo is a matter of

practicing, for example studying the example models we use, or reimplementing models discussed in publications.

Let's start with the implementation of a model of termites making piles of wood chips. You start a new NetLogo model and first create the interface. You add two buttons 'setup' and 'go'. The setup button is at the observer level and will be executed once. The go button is at the observer level and goes forever.

Also add two sliders: "number" with a minimum of 1 and a maximum of 300 with a step size of 1, and "density" with a minimum of 0 and a maximum of 100 with a step size of 1.

Next, you will go the "Code tab", and define setup as follows

```
to setup
  clear-all
  set-default-shape turtles "bug"
  ask patches
  [
    if random-float 100 < density [set pcolor yellow]
  ]
  create-turtles number [
    set color white
    setxy random-xcor random-ycor
    set size 5
  ]
end
```

This setup procedure randomly allocates wood chips (yellow patches) with a given density. The procedure also randomly put the white termites on the landscape.

Before you continue to click on the setup button in the interface. Does it work?

Now we continue with the go command. When you press "go", the termites (turtles), will continuously execute three rules:

- look around for a wood chip and pick it up
- look around for a pile of wood chips
- look around for an empty spot in the pile and drop off the

chip

```
to go
  ask turtles   [
    search-for-chip
    find-new-pile
    put-down-chip
  ]
end
```

The search-for-chip command is defined as follows

```
to search-for-chip
  ifelse pcolor = yellow
  [
    set pcolor black
    set color orange
    fd 20
  ][
    wiggle
    search-for-chip
  ]
end
```

Where the wiggle command is defined as

```
to wiggle
  fd 1
  rt random 50
  lt random 50
end
```

Find a new pile is defined as

```
to find-new-pile
  if pcolor != yellow
  [
    wiggle
    find-new-pile
  ]
end
```

Finally, drop off chip is modeled like

```
to put-down-chip
  ifelse pcolor = black
  [
    set pcolor yellow
    set color white
    get-away
  ][
    rt random 360
    fd 1
    put-down-chip
end
```

Where the command get-away is defined as

```
to get-away
  rt random 360
  fd 20
  if pcolor != black [get-away]
end
```

Click on the “go” button in the interface. What is happening? If you click on “Settings” and change the dimensions of the number of patches on the screen to a larger number, say 100×100 patches, will the same results appear?

Summary

This chapter provides an initial overview of NetLogo and instructions on how to start programming. Check out some of the many example models in the NetLogo library, check out what the primitives mean, and try to understand why the model produces the results it produces.

Some additional sources to learn more about programming in NetLogo are:

Videos by William Rand on Fundamentals of NetLogo

NetLogo Tutorial by Jen Badham

CHAPTER 5

Using AI for agent-based modeling

Key concepts

In this chapter, we will,

- – Discuss how to use LLMs to write NetLogo code
- – Use an LLM based to write an ODD
- – Explore the new field of GenABM

The increasing abilities, interest, and use of AI are changing many aspects of society, and that is also happening for agent-based modeling. The rapid developments will likely impact how we will develop, implement, and explore agent-based models in the future. Here I address three topics that are emerging at the time of writing this chapter.

Using LLMs to generate code

There is a difference between modeling and coding, as discussed earlier in this book. With modeling, we try to abstract a complex problem into a logical system of expressions based on theory and/or observational data. With coding, this abstraction can be implemented into computer code. Large Language Models (LLMs)

can assist with generating code, but I am skeptical about LLMs being able to create novel models.

Since coding is often a bottleneck for domain knowledge experts to do agent-based modeling, this development can make agent-based modeling more accessible. Suppose I am asking Claude, free version 1.15962.0, to create NetLogo code to simulate a population of agents randomly walking around a space, it will generate the relevant code:

My Prompt:

Can you generate NetLogo 7 code for a population of N agents who are randomly moving around in a spatial explicit environment?

Claude responded:

Here's a clean NetLogo 7 implementation with some useful features:

```
globals [
  step-size
]

turtles-own [
  energy
]

to setup
  clear-all
  set step-size 1.0

  create-turtles N [
    setxy random-xcor random-ycor
    set color one-of [red blue green yellow orange pink]
    set size 1.5
    set energy 100
    set heading random 360
  ]

  ask patches [
```

```
    set pcolor scale-color green (random 100) 0 150
  ]

  reset-ticks
end

to go
  ask turtles [
    wander
  ]
  tick
end

to wander  ;; turtle procedure
  ;; Randomly adjust heading
  rt random-float turn-angle – (turn-angle / 2)
  lt random-float turn-angle – (turn-angle / 2)

  ;; Move forward
  forward step-size

  ;; Optional: bounce off walls instead of wrapping
  if not wrap? [
    if xcor >= max-pxcor or xcor <= min-pxcor [
      set heading (180 – heading)
      forward step-size
    ]
    if ycor >= max-pycor or ycor <= min-pycor [
      set heading (- heading)
      forward step-size
    ]
  ]
end
```

to-report count-neighbors [radius]
* report count turtles in-radius radius – 1 ;; exclude self*
end
***Interface elements to add** (via the NetLogo GUI):*

Element	*Type*	*Settings*
N	*Slider*	*min: 1, max: 500, default: 100*
turn-angle	*Slider*	*min: 0, max: 180, default: 45*
wrap?	*Switch*	*default: On*
setup	*Button*	*—*
go	*Button*	*Forever*

What the model does:

- *Agents initialize at random positions with random headings and a random color*
- *Each tick, agents perturb their heading by a random amount within ±turn-angle, producing correlated random walks (more realistic than pure Brownian motion)*
- *wrap? toggle lets you choose between a toroidal world (agents wrap around edges) or a bounded world (agents bounce)*
- *The green-shaded patch background gives a sense of spatial heterogeneity, which you can later use to drive behavior (e.g., resource gradients)*
- *count-neighbors is a utility reporter you can call to check local density around any agent*

This is very useful information, but it also includes some elements that might not be relevant, like energy as an attribute for the turtles. You still need to know how to program in NetLogo but you can speed it up with generate code. The model above is

very straightforward and includes dynamics that are part of many NetLogo models. If you provide some non-standard description of a problem you like to model, LLMs likely create code, but it might not be meaningful to address the research questions you have.

In my ABM class, I typically give the assignment to replicate Figure 3 of Nowak and May (1992). This model is a deterministic cellular automata model on spatial cooperation, and the results are very sensitive to the precise implementation. When I provided Claude with the manuscript and asked it to write NetLogo code to replicate Figure 3, it was impressive that it provided the code, but it was not able to replicate the model's exact dynamics. Of course, this paper was selected for a replication assignment since it was not straightforward, but it also demonstrates that proper verification will be needed for code generated by LLMs.

A more advanced way of using LLMs is vibe coding, where a programmer relies heavily on AI coding assistants to generate, modify, and debug code using natural-language instructions rather than writing most of the code manually. Claude has Claude Code, and ChatGPT has Codex. Given the rapid development of those platforms, it is best to check out their training programs to get started with these tools.

Using Agentic AI to assist with writing an model documentation

The COMSES team created an AI based tool ODDER which will facilitates you to write a detailed model documentation using the Overview, Dynamics and Details (ODD) protocol following the 2020 ODD version. In a later chapter of this book we will discuss the ODD protocol in more detail.

This is work in progress, and at this moment it is not a simple plug-and-play to install it and work with it, but it will be in the future. You can also ask your favorite LLM to use the info in the specific GitHub folder to help you write your ODD, without having to install the Agentic AI on your computer.

Generative Agent-based modeling

With the rapid development of generative AI, we are starting to see it used to create agent-based models in which the agent's decision-making is the result of prompts to an LLM (Gao et al., 2024; Anthis et al., 2025). Various computer scientists are creating platforms to do those multi-agent simulations, where individual agents follow prompts and are created to represent persona. Within the private sector some startups emerged like https://societies.io/ and https://aaru.com/ who make simulations for their corporate clients.

There are, however, a number of limitations of generative ABMs like the difficulty of replication, the lack of validation of LLM-based human behavior (we know LLMs have biased as represented in their training data), and the fact that LLMs are black-boxes. When one develops generative ABMs, it might be that one is studying more the mysteries of LLMs instead of crafting models in line with the modeler's understanding of the world. It is too early to say how this new approach will be a productive scientific instrument improving our understanding of societal dynamics.

Summary

There are new exciting possibilities emerging with LLMs in combination with agent-based modeling. The coming years will tell what the useful practices and how agent-based modeling will change as a tool for academic research are.

References

Anthis, J.R., Liu, R., Richardson, S.M., Kozlowski, A.C., Koch, B., Brynjolfsson, E., Evans, J. and Bernstein, M.S., 2025, April. Position: Llm social simulations are a promising research method. In *Forty-second International Conference on Machine Learning Position Paper Track*.

Gao, C., Lan, X., Li, N., Yuan, Y., Ding, J., Zhou, Z., Xu, F. and Li, Y., 2024. Large language models empowered agent-based modeling and simulation: A survey and perspectives. *Humanities and Social Sciences Communications*, *11*(1), pp.1-24.

Nowak, M.A. and May, R.M., 1992. Evolutionary games and spatial chaos. N*ature*, *359*(6398): 826-829.

PART II

GETTING STARTED WITH SOME CLASSICS

In this part of the book, we will get you more familiar with concepts of modeling complex adaptive systems and the NetLogo language by discussing the implementation and outcomes of a series of classic agent-based models. We will illustrate those models by some explorations and adjustments of the models you can do yourselves to derive some new insights and understanding of various types of systems.

CHAPTER 6

Cellular Automata

Key Concepts

In this chapter, we will:

- discuss cellular automata and their applications,
- explore the emergence of complex patterns from simple rules,
- see the difference between sequential and non-sequential updating.

Originally, the cellular automata (CA) approach was introduced by John von Neumann and Stanislaw Ulam during the 1940s and 1950s at the Los Alamos National Laboratory. A cellular automata is a system with an environment of cells that can have a limited number of states. Time goes by in discrete steps. According to some (deterministic) rules set by the researcher, which are the same for each cell, the state of a cell at the next time step depends on its present state and the states of all its surrounding cells at the present time step. The resulting surprisingly complex dynamics that evolved from this simple set of rules have led scholars in many disciplines to study the complex dynamic behavior of systems since

the early 1970s using the CA approach. The essential properties of a CA are:

· a regular n-dimensional lattice (in most cases, n is one or two dimensions), where each cell of this lattice has a discrete state; and

· a dynamical behavior, described by the so-called rules. These rules describe the state of a cell at the next time step, depending on the states of the cells in its neighborhood.

The fundamental element of a CA is the cell that is represented by states. In the simplest case, each cell has the binary state equal to 1 or 0, which may represent cooperate/defect in a social dilemma, adopt/not adopt a product, or build environment/nature in terms of land use. In more complex simulations, the cells can have multiple states. These cells are arranged in a lattice. The most common CAs are built in one or two dimensions. The cells can change the state by transition rules, which determine the state of the cells for the next time step. For example, if the cell with the highest payoff in the neighborhood in the last time step was a cooperator, cooperate in the next time-step.

Another example is for a cell to adopt an innovation if X percent of the neighboring cells have also adopted the product. In cellular automata, a rule defines the state of a cell as a function of the neighborhood of the cell. The neighborhood defines the sphere of influence of the cells, and the type of neighborhood can affect the results of the simulations. The most common neighborhoods for two-dimensional CA are given in Figure 1.

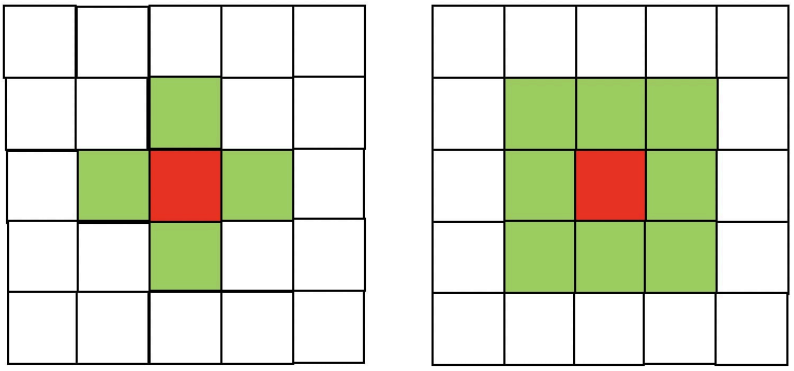

Figure 1. Examples of cellular automata neighborhoods. Left is the Von Neumann Neighborhood and Right the Moore Neighborhood. The red cell is the center cell, the green cells are the neighborhood cells. The states of these cells are used to calculate the next state of the (red) center cell according to the defined rules.

CAs have been very successful as theoretical models of complexity and have been applied to diverse topics such as land-use change and cancer modeling. In the remainder of this chapter, we will explore some classic cellular automata models you can also find in the NetLogo libraries.

Game of Life

John Conway developed the cellular automata Game of Life in 1970. Each patch can have the state "dead" or "alive." Each tick the state of all patches are updated sequentially. The state of the cell depends on the state of patches in the Moore neighborhood and its own state.

We will look at the model "Life" in the NetLogo library "Computer Science/Cellular Automata." The rule that defines the state of the patch is implemented in the NetLogo model as

```
ifelse live-neighbors = 3
       [ cell-birth ]
       [ if live-neighbors != 2
         [ cell-death ] ] ]
```

Where live-neighbors was first calculated as

```
set live-neighbors count neighbors with [living?]
```

The rule says that if there are 3 neighbors alive, the patch is alive, and if there are 2 neighbors alive, the state of the patch remains the same as the previous tick. In all other cases, the state of the patch is dead.

The intuition for those rules is that an alive patch stays alive if there are not too many other alive patches around (overpopulation) and not the little (underpopulation). And a dead patch comes alive if there is the right amount of alive patches (=3) in the neighborhood.

Now try out the model (For NetLogo web version click here). You can start with an initial random distribution of alive and dead patches. You see a continuous change of the states, and it looks like the movement of patterns on the screen. This is an example of an emergent phenomenon since the rules define only the state of individual patches, not the larger-scale outcomes.

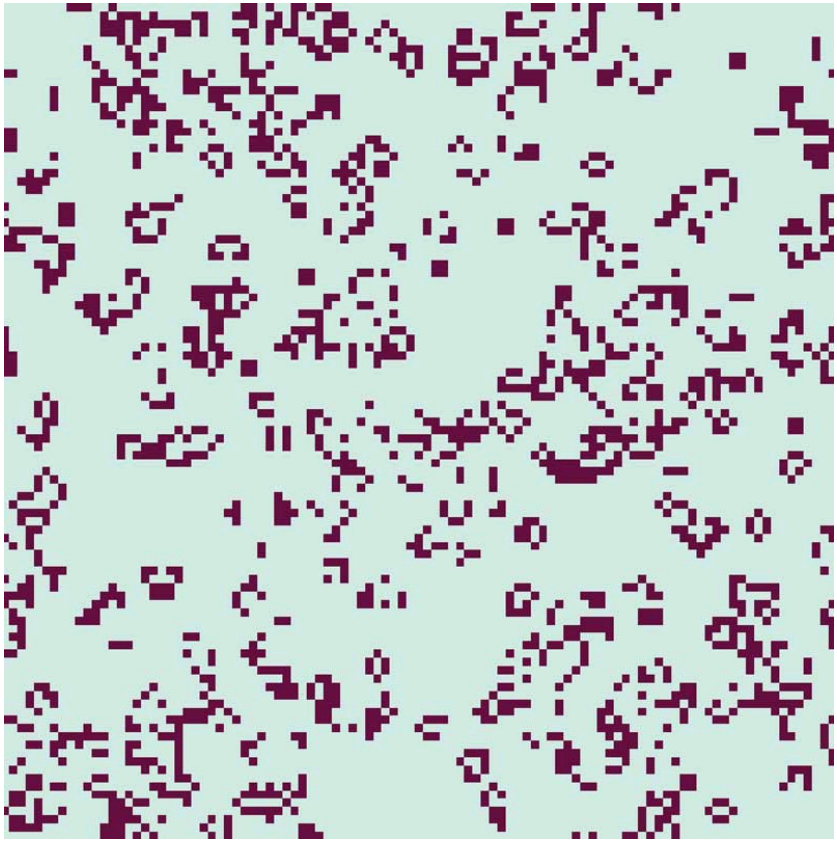

Figure 2. Snapshot of the game of life simulation.

There are many elements to explore with the model. One example is to vary the initial density of alive patches. Are there ranges of densities of initial alive patches for which the system will die out?

We will now use the Life model as a way to demonstrate the importance of the timing of updating information. In procedure go you see that the patches are updated according to the following lines of code:

```
ask patches
  [ set live-neighbors count neighbors with [living?] ]
```

```
ask patches
  [ ifelse live-neighbors = 3
    [ cell-birth ]
    [ if live-neighbors != 2
      [ cell-death ] ] ]
```

One may think this could be simplified, like

```
ask patches
  [ set live-neighbors count neighbors with [living?]
   ifelse live-neighbors = 3
    [ cell-birth ]
    [ if live-neighbors != 2
      [ cell-death ] ] ]
```

What do you think would happen and why? Try it out!!

What you will see is the difference between sequential and non-sequential updating. In the original implementation, all patches were updated using the state of the patches of the previous tick. However, in the new implementation, this is no longer the case since the state of neighboring patches might be updated before you calculate live-neighbors for the patch you are updating. As a result, you get very different outcomes.

What is the correct way of updating? That depends on the system you try to simulate. The game of life was based on sequential updating, and you do not get the same interesting outcomes if you use non-sequential updating. We will explore the consequences of other types of updating sequences in other examples in the book.

Elementary Cellular Automata

The simplest possible cellular automaton can be considered to be in one-dimension, where the patch has only a neighbor to the left and to the right and has two different states 0 and 1. Now we can define eight possible neighborhoods:

000
001
010

100
011
101
110
111

Each sequence of 3 numbers refers to the left neighboring patch, the patch for which the state is evaluated, and the right neighboring patch. For each of the 8 possible situations, we can define the outcome being 0 or 1. For example, 000 -> 1 means that a patch with the state 0 and two neighboring patches being zero, the new state, in the next tick, will be 1.

We define a rule as the outcome for each possible neighborhoods. For example, the so-called Rule 90 is defined as

current pattern	**111**	**110**	**101**	**100**	**011**	**010**	**001**	**000**
new state for center cell	0	1	0	1	1	0	1	0

Since there are eight possible neighborhoods and 2 possible states, there are possible rules. Stephen Wolfram explored the consequences of the 256 rules systematically, and he found interesting emergent outcomes for some of them. In fact, Wolfram identified 4 different classes when the initial states of the patches are randomly defined.

Class 1: Rapid convergence to a uniform state.

Class 2: Rapid convergence to a repetitive or stable state.

Class 3: States remain to appear randomly.

Class 4: Formation of structures and patterns.

In the NetLogo Library Computer Science\Cellular Automata, you find the model CA 1D elementary. Click on setup-random and then on go, and you get something like the following:

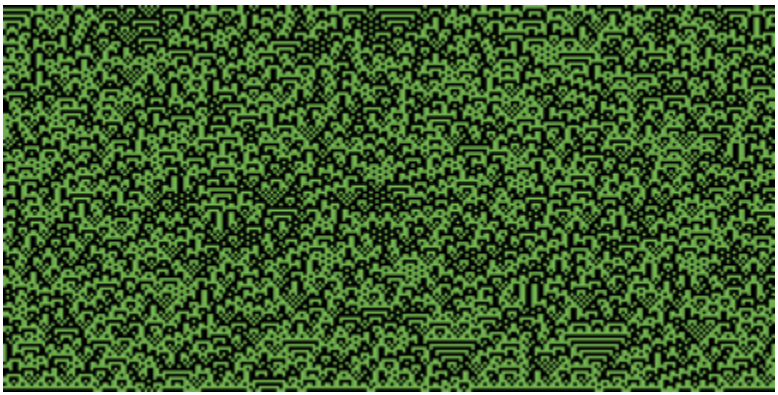

Figure 3. Snapshot of an elementary cellular automaton simulation.

How to read this image? The top row consists of the original random states of green and black patches. The second row is the outcome of the patches' states after applying the rule to the states of the patches of the first row. Like the Game of Life, the computation of the states is sequential, meaning that all patches of the second row, depending on the rule, apply to the patches' states in the first row. Running the model leads to more rows being calculated, and the figure you see above is a figure of the patches' states over time. Dependent on the rules, you find different patterns. Try out some rules by changing the slider "rule."

One interesting rule is rule 110, which leads to the evolving pattern in Figure 4. This example demonstrates why some scholars study cellular automata since it provides examples of simple rules that lead to complex patterns we find in nature. Compare Figure 4 with the pattern on the shell in Figure 5. How could nature produce such complex patterns with probably simple rules encoded in genetic information? The elementary cellular automata provide examples of possible mechanisms.

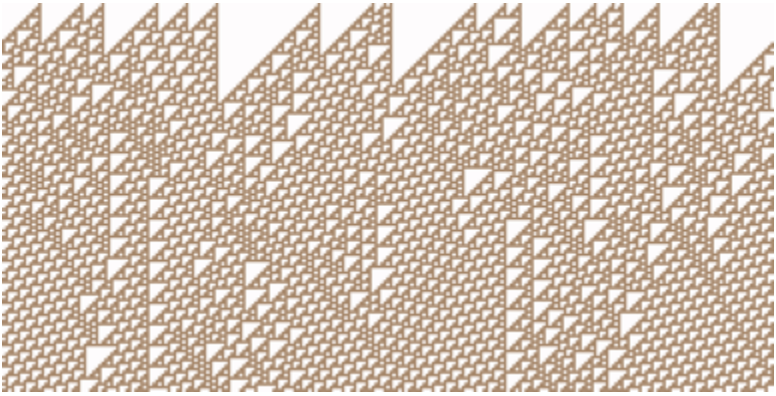

Figure 4. Rule 110

Figure 5. Pattern on a shell in the hand of the author.

As you can imagine, there are many variations possible of the elementary cellular automaton, by changing the number of states, and the kind of rules possible. In the model "CA 1D Totalistic" from the NetLogo library "Computer Science/cellular automata," we see an example of 3 states (3 colors), which lead to more than 2000

possible rules. This shows that the set of possible outcomes quickly starts to become very big once we go beyond the simplest possible case. Figure 6 shows an example of the 3 states' version.

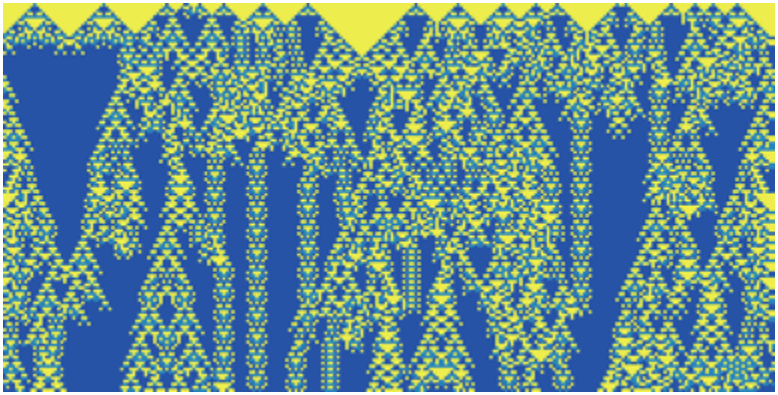

Figure 6. Screenshot of simulation with 3 states.

A final example in this section is the model "CA stochastic," which you can also find in Computer Science/Cellular Automata. In this case, the rule is stochastic. Given the states of the 3 patch neighborhood, there is a probability that a patch will be in one state or the other. This leads to a much broader spectrum of possible outcomes, and Figure 7 demonstrates one example.

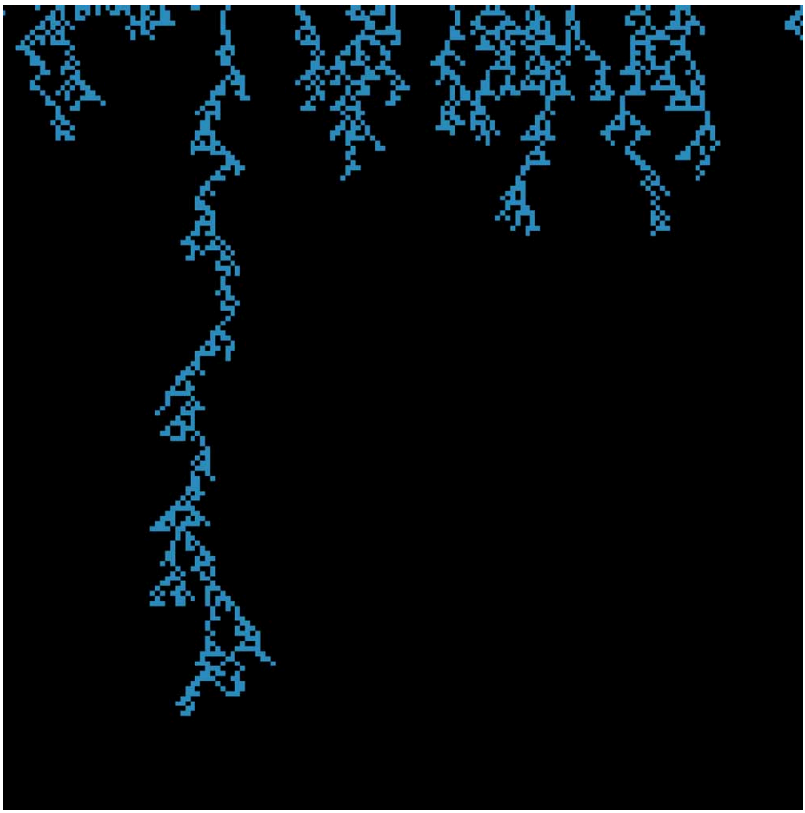

Figure 7. Screenshot of a simulation of a 1-dimensional 2-state cellular automaton with stochastic rules.

Fire Model

The final example in this chapter relates to land cover change, a typical application of cellular automata. In 1992 Drossel and Schwabl published a cellular automata of a forest that is confronted with a fire at one side of the forest. With a few simple rules, they model the spread of the fire. How far will the fire spread? What would be a density of trees for which there is a higher probability of fires? The work of Drossel and Schwabl is done in the context of percolation theory, which we will not dive into in this chapter.

Sufficient to say is that there is a threshold of trees' density for which the risk of forest fires dramatically increases.

So what are the rules of the model? We will use the rules as used in the model Fire that you can find in the NetLogo library Earth Science (You can also access the Netlogo Web version here). Although the NetLogo model uses turtles to simulate the spread of fire, the model can be reprogrammed as a strict cellular automaton (using patches only). Try out to reimplement the Fire model with patches only.

A patch can have three states. a tree (T), a burning tree (B), or be empty (E). Given fire at the left side of the screen, a patch change states in the following way.

– B[t] -> E[t+1]

– T[t] -> B[t+1] if there is a B[t+1] in the Von Neumann neighborhood.

This rule suggests that if there are big spaces between trees, and therefore becoming likely that a Von Neumann neighborhood has no trees, the fire will not spread. What would be a safe density of trees? We can explore the model by running it with different densities. In Figure 8, you see results for densities of 57%, 60%, and 63% of the patches having trees. The results suggest that there is a major change in the results when we have some small changes in the density.

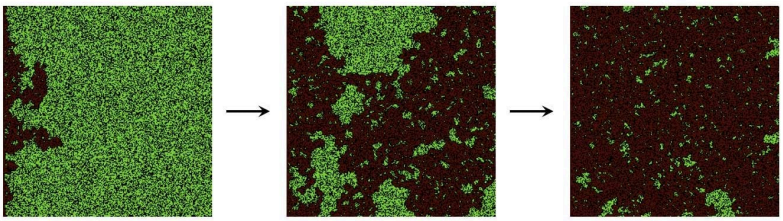

Figure 8. Examples of the outcomes of the Fire model with densities 57%, 60%, and 63%.

If we simulate the model 100 times for each density level, we get

the following figure of the percentage of the trees being burned (Figure 9). We see that indeed there is a sudden change in the percentage of the trees being burned. This threshold is an emergent phenomenon based on the assumptions of the model: randomly allocating the trees, and the rules of the spread of the fire.

Things to explore with the model will be to see how this threshold will change if trees are not randomly allocated on the landscape, but including fire breakers (rows of empty patches). And what if the rule of fire spread includes more complexity such as a probability of further spread of the fire due to wind flowing burning branches over the landscape.

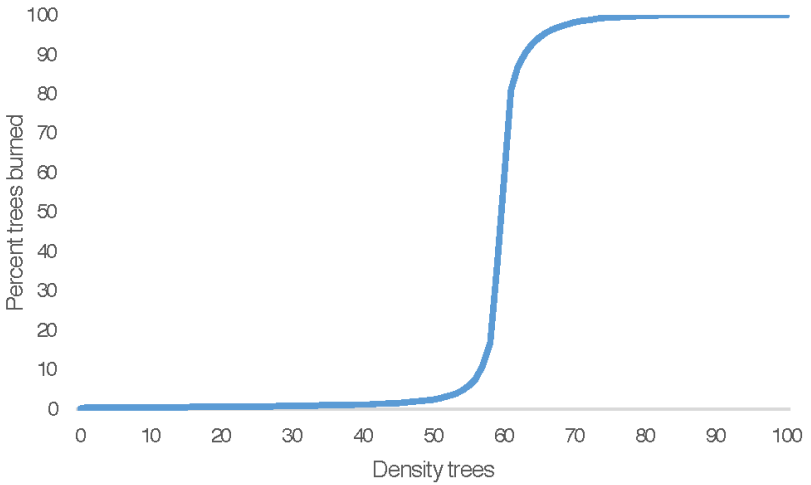

Figure 9. Results of simulating the Fire model 100 times for each density level.

To conclude this chapter, we hope you have seen that simple rules could lead to complex dynamics of systems, and with models, we can systematically explore the consequences of model assumptions and mechanisms (rules). In the next chapters, we move back to traditional agent-based models, where we include

agents who may have different rules, may move, and have different types of neighborhoods of influence.

Summary

Cellular automata are the simplest type of agent-based models by using a landscape of regular patches and provide simple rules that hold for each patch. Those rules use local information to define changes among the discrete number of states. Those local rules could lead to surprisingly comprehensive macro-level patterns.

References

Drossel, B. and Schwabl, F. (1992), Self-organized critical forest-fire model. *Physical Review Letters* 69, 1629–1632.

Gardner, M. (1970) Mathematical Games – The fantastic combinations of John Conway's new solitaire game "life". *Scientific American* 223 (4): 120–123.

Wolfram, Stephen (2002). A New Kind of Science. Wolfram Media, Inc

CHAPTER 7

Social Dynamics

Key Concepts

In this chapter, we will:

- meet some social scientists who have developed foundational work that shaped ABM,
- see that micro-level motivations lead to macro-level behavior,
- show that selfish rational individuals can cooperate in social dilemmas,
- demonstrate that a diversity of strategies can lead to good outcomes when we have limited information.

In this chapter, we discuss a sample of classic agent-based models in the social sciences. In fact, at the time of publication, those models were not called agent-based models, and some were even done by hand with pennies and dimes. Those examples illustrate that formal agent-based thinking was around within the social sciences for a long time before the tools to implement them became more widely available. The examples chosen also allows

for the introduction of some key figures in the social science that have been influential to the agent-based community.

Segregation

Thomas Schelling (1921-2016) was an economist who won the 2005 Nobel Memorial Prize in Economic Sciences for his work on conflict and cooperation through game-theory analysis. However, within the agent-based modeling community, he is seen as one of the founders of agent-based computational economics, although he did not use computer simulation.

A classic book of Schelling relevant for agent-based modeling is "Micromotives and Macrobehavior," published in 1978. In that book, Schelling used logical reasoning to explore the outcomes of individual behaviors leading to emerging macro-level patterns, such as patterns of seats being occupied in theaters, to the spatial distribution of housing prices. The later is an example of the role of competition when there are a limited number of options. If parents want to live in neighborhoods with the best schools, there will be a bidding war for spots in those neighborhoods, increasing the prices of those schools.

The most well-known example of Schelling's work on micro motives and macro behavior is the segregation model, initially published in 1969 and 1971. Do we observe the segregation of distinct groups in cities because people do not want to live next to those who are different? Schelling shows that a preference for living in a mixed neighborhood to a specific limit still can lead to total segregation. Segregation can be caused by a slight preference to live next to similar neighbors without being unhappy when living in neighborhoods dominated by those who are different. This surprising result was initially implemented using coins on a graph paper and using pennies and dimes instead of agents.

Nowadays, the segregation model is a classic model in agent-based modeling software packages. In NetLogo, we can find this model in the Social Science folder. The model considers agents randomly allocated on the patches, one per patch, where each

agent has one of two possible colors. There are a number of empty patches available. Each tick agents define whether they are happy. If they are not happy they move to an empty patch. The agents' updating is that they first determine whether they are happy, and then move all agents who are defined unhappy to empty patches.

What defines whether an agent is happy? An agent is happy if the number of turtles on the eight patches around an agent which have the same color as the color of the agent itself if above a certain threshold %-similar-wanted. This is calculated as follows. First, the number of agents nearby with the same color and the number of agents with a different color is defined. Then it is checked whether the percentage similar observed is above the threshold.

```
set similar-nearby count (turtles-on neighbors)  with [ col
set other-nearby count (turtles-on neighbors) with [ color
set total-nearby similar-nearby + other-nearby
set happy? similar-nearby >= (%-similar-wanted * total-near
```

Suppose we start with a random distribution of agents, where 50% of the neighbors have the same color and assume that the threshold %-similar-wanted is 30%. What would happen if unhappy agents would move, and we simulate until there are no unhappy agents? It might be surprising that this leads to a similarity percentage of 74% and a segregated society. Why is this happening while agents are happy if the majority of the neighbors are different? The reason is that some agents have less than 30% similar initially because those agents move to new locations, increasing the level of similarity between agents. Even though most agents were initially happy, moving those who were unhappy increases the level of segregation a lot.

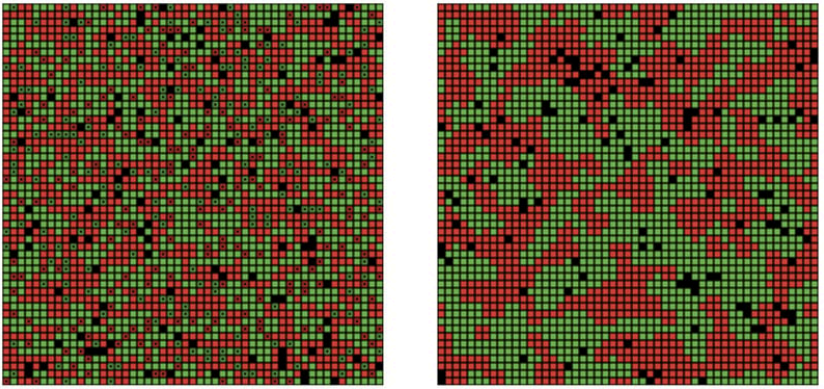

Figure 1. Initial and final states of a simulation of the segregation model with a similarity threshold of 30%.

If we vary the density of the agents on the map and the threshold for agents to be happy, we see that a similarity threshold of 30% leads to a macro-level similarity of more than 70% independent of the density of the agents (Figure 2). When we increase the similarity threshold, the aggregated similarity keeps slowly growing, except for simulations with a high density of agents, and unhappy agents keep moving around without finding a happy spot.

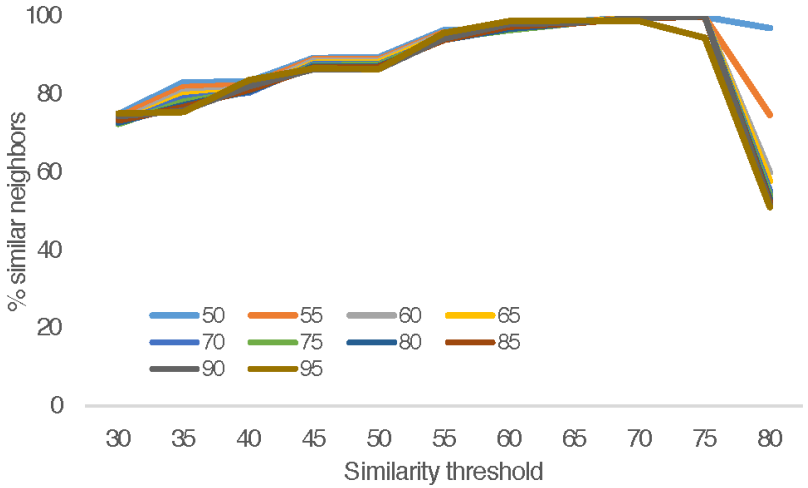

Figure 2. Relationship between the similarity threshold of agents, and the macro-level similarity of neighbors for different densities of the agents on the map (50%-95%). Results depicted are the mean for 100 simulations for each parameter setting.

Things you can do with the Schelling model. Add a histogram and see how the distribution of similarity change over time. What happens if agents have a bigger neighborhood than the 8 neighbors impacting their happiness (for example, use a radius and explore how a bigger radius impact the outcomes)? What happens when there are more than 2 groups? What if agents have different thresholds that define happiness?

The model is also an excellent example to explore the consequences of different ways of updating agents. In the current version, agents first define all whether they are unhappy or not. Then all unhappy agents move, even though the earlier movement of agents during that routine could make agents that we defined unhappy happy again. Hence an option would be to include turtles' movement in the procedure of updating the happiness of turtles. What would be the consequences?

If you use ask, the updating would be in random order. But one

could also imagine that most unhappy agents are most eager to move. So what if you use sort-on to allow agents who are most unhappy to move first to a new location.

Cooperation

A key problem studied by social scientists is social dilemmas where there is a conflict between the interest of the individual and the interest of the group. An example is doing group projects for your class. Although everyone in the group would benefit if everyone put in a lot of effort into the group project to get the best outcome, each individual would benefit from free-riding on the effort of others. Will this lead to a disaster for all group projects, or do we find groups who work very well together? There are many of those social dilemmas in everyday life, from standing in line for the cashier to paying taxes, getting vaccinations, and correcting articles on Wikipedia.

A classic game in the social dilemma literature is the Prisoners' dilemma game. The game has initially been constructed by mathematicians Merrill Flood and Melvin Dresher at RAND corporation in 1950. Albert Tucker formulated the model formally and gave it the title prisoners dilemma game, where the 2 players were members of a criminal gang who had to decide to cooperate with the police or not and where prison length were the payoffs. We discuss the game in more general terms.

Given are two players, agent 1 and agent 2, who have to decide to cooperate or defect. There is no communication between the agents, and they only consider the material payoff. If both players cooperate, they both receive a payoff of 3 units. If both players defect, they both receive a payoff of 1 unit. However, if one player cooperates and the other defects, the cooperator gets nothing, and the defector receives 5 units. Hence selfish rational agents will defect in this game. This is called a Nash equilibrium, named after John Nash. In Table 1 we provide the payoff matrix.

Agent 1 \ Agent 2	C	D
C	{3,3}	{0,5}
D	{5,0}	{1,1}

When this game is performed with humans where the payoff is money, we do not find that all players defect all the time. In fact, we would not have an advanced human society if humans would mainly defect in social dilemmas. There are many psychological reasons why humans do not behave like psychopaths but have feelings, moral values, follow social norms, communicate, experience pleasure from cooperation, etc.

We will not dwell on the psychological explanations here but look at this problem from a modeling perspective. If people interact with others in social dilemmas, what are the conditions for which we see cooperative outcomes? We will illustrate this with the model "Prisoner's Dilemma N-Person Iterated" in the Netlogo library folder Social Science/Prisoner's Dilemma. In this model, agents wander randomly around, and each tick an agent tries to find a partner in a neighboring patch. If an agent is partnered up, the agent plays a Prisoner's Dilemma game. Each agent has a strategy that could take into account that they meet up again, and they may keep records on previous encounters.

For simplicity's sake, let's have 50 agents who always cooperate and 50 agents who always defect. When we run the model, we see that the average payoff for defectors is around 3, and the average payoff for cooperators is about 1.5 (Figure 3). This is not surprising since agents will bump into a cooperator or defector by equal chance, and a cooperator will receive 3 (with a cooperator partner) or 0 (with a defector partner), and the defector will receive 5 (with a cooperator partner) or 1 (with a defector partner).

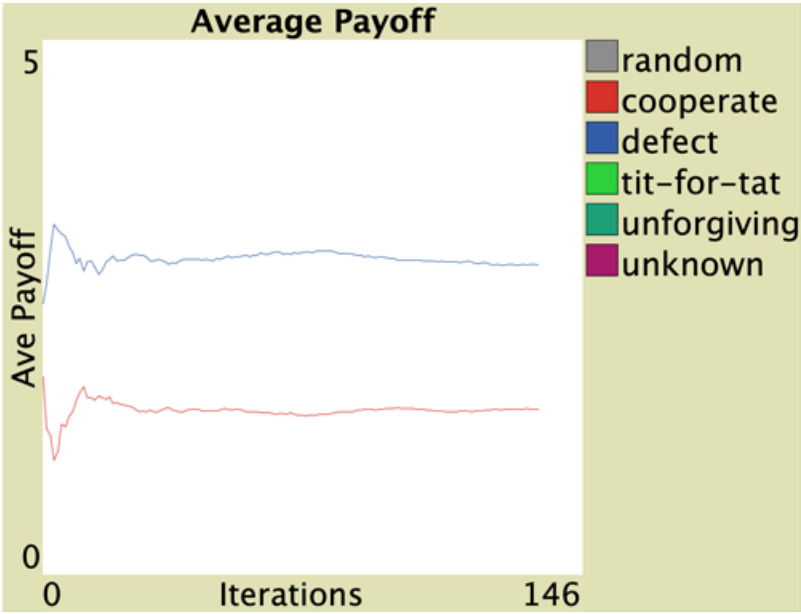

Figure 3. Results of a simulation of 100 agents playing Prisoner's Dilemma games.

In 1980 Political Scientist Robert Axelrod published two papers with the results of a tournament. Dr. Axelrod invited colleagues to submit an algorithm for a player who will, in a computerized tournament, play the Prisoner's Dilemma game for 200 rounds with each other player, as well as the own strategy and a strategy of randomly deciding to cooperate or defect. The winner of the first round was the mathematical psychologist Anatol Rapoport. The strategy, Tit for Tat, was the simplest submitted and worked as follows. Start with cooperation, and if the partner cooperated in the previous round, cooperate, and if the partner defected in the last round, defect. In this way, the strategy starts nice, rewards cooperation, and punishes defection. A second round of the tournament was played where people knew which strategy won in the first round, invited contributions from a broader audience, and did not have a fixed number of 200 rounds, but a probability that

each round could be the last round. Again Tit for Tat was the best strategy.

Another strategy that was doing well was called "unforgiving," where an agent starts with cooperation but will switch to defection once the partner defected once. The strategy of Tit for Tat and unforgiving look the same. The agent checks whether the partner defected by checking the information in the partner history list. If the partner is listed as a defector, the agent will defect, while if the partner is not listed as a defector, the agent will cooperate.

```
set partner-defected? item ([who] of partner) partner-histo
  ifelse (partner-defected?) [
    set defect-now? true
  ] [
    set defect-now? false
  ]
```

The real difference between the strategies is how they update their history of their partners' decisions. In the initialization, both agents have falsely listed for each possible partner. Once a tit for tat agent played a game, the history will be updated along with the decision of that game. In contrast, the unforgiving agent updates the history if the partner defected, hence being unforgiving.

```
to tit-for-tat-history-update
  set partner-history
    (replace-item ([who] of partner) partner-history partne
end

to unforgiving-history-update
  if partner-defected? [
    set partner-history
      (replace-item ([who] of partner) partner-history part
  ]
end
```

We now run the model with equal numbers of random, cooperator, defector, tit for tat, and unforgiving. We see that the unforgiving strategies lead to higher average payoff than tit for tat (Figure

4). Random strategies have the worst performance. Why do we see unforgiving being the best strategy while tit for tat won the two tournaments? Is tit for tat, not the best strategy? Well, that is not so simple to answer. The best strategy depends partly on the strategies submitted by other players. In Figure 5 we see that if we remove the random strategy, the tit for tat strategy and the unforgiving strategy start to become equal in performance.

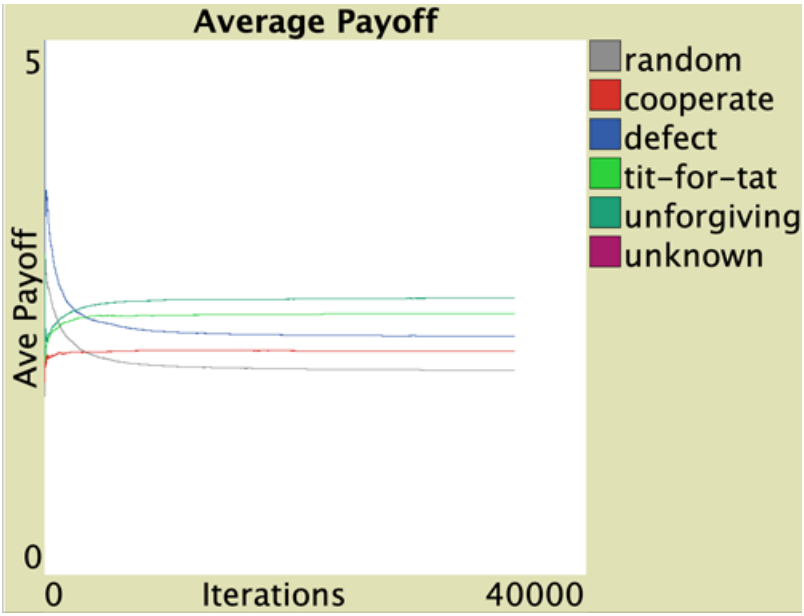

Figure 4. The average payoff for the 5 strategies (unknown is not included) over time.

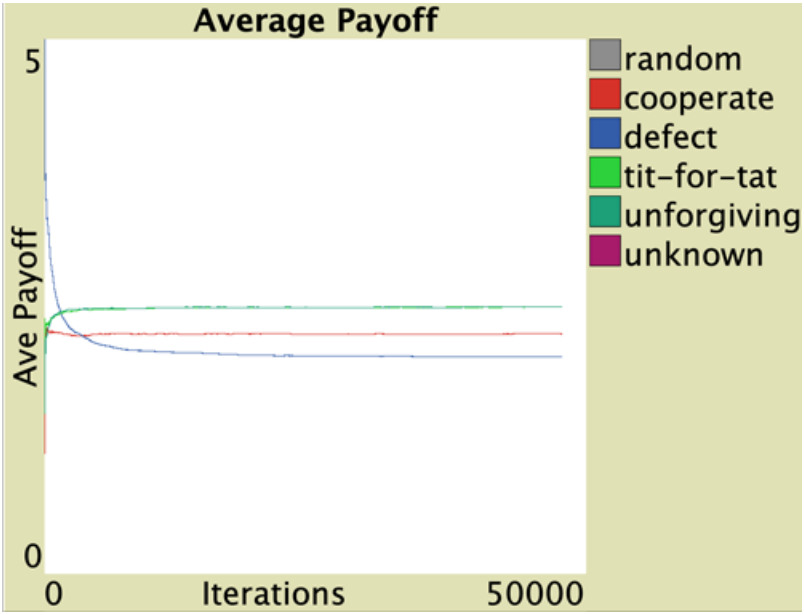

Figure 5. Like Figure 4 but now excluding the random strategy.

Axelrod published in 1984 his classic book "The Evolution of Cooperation," in which he starts exploring all kinds of variations of strategies in different forms agents could play. This led to a whole field of investigation of thousands of publications where computer simulation was used to explore how cooperative strategies could evolve or persist in various types of games, with different spatial interaction, competition, and evolution of strategies, different types of information available to the players, etc.

With the model we have used in this example, you could explore some interesting variations too. What if agents without partners keep moving towards those who have good records and away from those who have bad records? What agents can share information about their history of other agents?

The El Farol Problem

Brian Arthur is an economist who has been instrumental in connecting complexity theory with economics. He is primarily

known for his study of the impacts of positive feedback in economics, which shows how small random events could affect which products dominate the market. We will discuss this in a later chapter. In this chapter, we will look at a problem Brian Arthur formulated in early 1994, which is now known as the El Farol Bar problem.

The main gist of the problem is that people have to make decisions under uncertainty, while we often still see a certain order in the results. El Farol is a bar in Santa Fe, New Mexico, USA, which is the inspiration for the problem. Given is a finite population N. Every Thursday night, all N agents decide to go to the El Farol Bar. Since El Farol is quite small, it will not be fun if everyone comes, it will be too crowded. Therefore, the preference of all agents are defined as:

If the share of the population at the bar is less than 60%, then they are happy, while if the percentage of the population at the bar is more than 60% of the agents are unhappy.

Note that the problem was formulated in the early 1990s, a time before mobile phones, and the assumption is that all agents make decisions at the same tie to come or not. If all agents have the same deterministic strategy, they all will end up at home. Since it is all at the bar or all at home. One can define stochastic strategies, say agents flipping coins, that could lead to an attendance that makes agents in the bar happy most of the time. The model in the NetLogo library (El Farol in the Social Sciences folder) demonstrates what would happen if agents have deterministic strategies, but not all agents have the same strategy.

A strategy is a model to predict attendance. Based on the predicted attendance, more than 60% or not, the agent will decide to go or not. The prediction is based on observations of past ticks, and all agents have the same information of past attendance but may use different strategies to predict attendance for future rounds. The strategy is formulated as

p(t) = c * 100 + x(t – 1) * a(t – 1) + x(t – 2) * a(t -2) +.... + x(t – MEMORY-SIZE) * a(t – MEMORY-SIZE)

Agents are assumed to have a number of strategies, number-strategies, to predict the future. At the beginning of the simulation, all agents generate number-strategies initial strategies like

```
set strategies n-values number-strategies [random-strategy]
```

where n-values generates a list of length number-strategies containing values computer by repeatedly running the reporter random-strategy. In random-strategy, the weights of each of the strategies are randomly generated between -1 and 1.

```
to-report random-strategy

  report n-values (memory-size + 1) [1.0 - random-float 2.0

end
```

During each tick, the agent checks its own sample of strategies, to see which strategies best predicts the observed historical pattern. The historical observations are compared with each strategy in the portfolio of the agent, and the best strategy is the strategy with the best fit with the historical observations:

```
repeat memory-size [
   set prediction predict-attendance the-strategy sublist h
   set score score + abs (item (week - 1) history - predict
   set week week + 1
]
```

When the agent makes a prediction, they use the best strategy that explained the past and use this to make a prediction about the future:

```
set prediction predict-attendance best-strategy sublist his
```

where sublist creates a sublist of the list history as input for the reporter predict-attendance.

```
to-report predict-attendance [strategy subhistory]
```

With this model, we can explore the benefit of more models in your portfolio, and more historical data in your memory is. Will more data and models lead to better performance? We simulated 1000 runs for 100 ticks for a number of different memory sizes and several models and looked at the attendance at the bar at tick 100. For each simulation, we check whether those who went to the bar were happy. Figure 6 shows that if agents have only one strategy, bar-goers are not happy due to overcrowding. When we have about 5 strategies and a memory of 7 to 10 ticks, we see the best performance of 80% of the time agents in the bar is happy. Interesting is that more models are not necessarily better. This might be explained by the observation that with more models, the best strategy is the one who best demonstrated the past and with many models to fit, this may lead to switching the whole time between models who best explained the past but are not necessarily be robust predictors for the future.

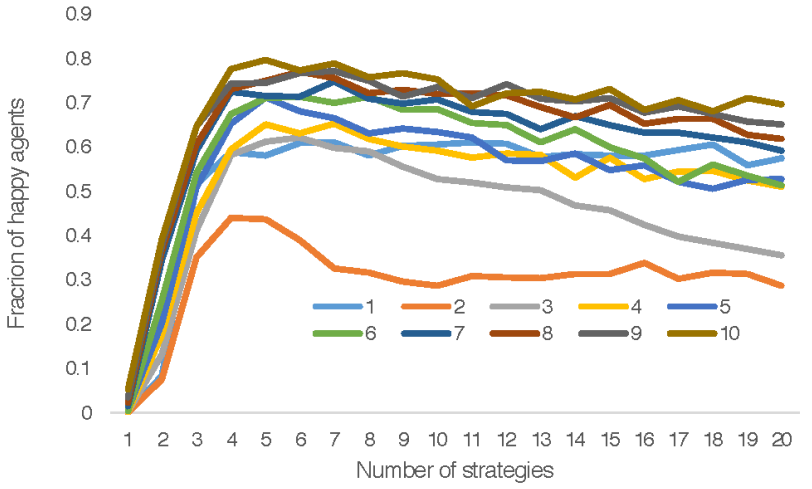

Figure 6. Fraction of simulations of happy bar-goers dependent on the number of strategies and the length of memory.

There are many variations to explore. What if agents who do not go

to the bar, do not have perfect information about the attendance at the bar last week? What if agents with their friends, assuming agent are in a social network, can share their intentions to go to the bar, will this improve predictions? What if not all agents go to the bar even though they think it could be great since something came up? Check out a variation of the model and let me know what you find.

Summary

In this chapter, we discussed a sample of classic models of social dynamics. In all those models, an agent's decision rule depends on the state of the agent (happy or not) and the aggregated state observed about others (number of neighbors. number of cooperators, number of bar visitors). The agents all had simple reactive rules to local cues leading the macro-level patterns of social relevance.

References

Arthur, W.Brian (1994) Inductive Reasoning and Bounded Rationality. American Economic Review 84: 406–411.

Axelrod, Robert (1980) Effective Choice in the Prisoner's Dilemma, Journal of Conflict Resolution, 24: 3-25.

Axelrod, Robert (1980) More Effective Choice in the Prisoner's Dilemma, Journal of Conflict Resolution, 24:379-403.

Axelrod, Robert (1984) The Evolution of Cooperation (New York: Basic Books).Schelling, T.C. (1969) Models of segregation, American Economic Review, 1969, 59(2), 488–493.

Schelling, Thomas C. (1971) Dynamic Models of Segregation, Journal of Mathematical Sociology, 1(2), pp. 143–186.

Schelling, Thomas C. (1978) Micromotives and Macrobehavior, Norton.

CHAPTER 8

Movements and Patterns

Key Concepts

In this chapter, we will:

- see how small changes in rules of movement lead to very different macro-level patterns,
- learn that the same micro-level rules can reproduce observed spatial patterns of different species,
- observe that agents adjusting the environment is one way of communication and create emergent patterns.

In this chapter, we look at a variety of models where agents move around, leading to emergent spatial patterns. In those models, the agents will move at a certain speed and decide where to move next. The examples show the different consequences of how to make decisions when to move where. Decisions can relate to current, and past positions of others observed directly or via environmental cues.

Flocking of Boids

Flocks of starlings or schools of fish have mesmerized many of us with their complex dynamic patterns. How do those animals

choreograph those complex dynamics? How do they create those patterns without having a choreographer or leader who directs what to do when? Can we simulate those patterns using simple rules only?

Biologists have been using computer simulations to understand the formation of flocks of birds and schools of fish. Computer graphics expert Craig Reynolds, who works in the Hollywood film industry, demonstrated in 1986 a simulation model of artificial birds, boids, which behaved as flocks of birds using only 3 simple rules.

All boids have the same speed and make each tick a decision to adjust its direction. The adjustment of direction is the result of 3 decision rules, namely separation – adjust direction to avoid bumping into nearby flockmates, alignment, adjust direction toward the average heading of local flockmates, and cohesion, adjust direction to move towards the center of the local flockmates.

To see how this can be implemented in Netlogo, we look at the model Flocking in Netlogo library Biology.

For an agent, we first identify the flockmates as the agents within a certain radius:

```
set flockmates other turtles in-radius vision
```

If the agent has flockmates, it may change its direction. First, it checks whether the closest flockmates are within a minimum-separation distance. If this is the case, the agent will call the procedure separate, where the agent will adjust the heading away from the nearby flockmate. If the agent is not close to bumping into another agent, it will align and cohere. Align is adjusting the heading towards the heading of the average heading of flockmates, and cohere is to adjust the heading towards the center of the flockmates group. Each of the 3 adjustment procedures has a maximum degree that may change during one tick to avoid sudden big adjustments.

```
if any? flockmates
    [ set nearest-neighbor min-one-of flockmates [distance
      ifelse distance nearest-neighbor < minimum-separation
        [ separate ]
        [ align
          cohere ] ]
```

When we run the model, we see that the random-looking agents will form flocks. You can access the NetLogo Web version here.

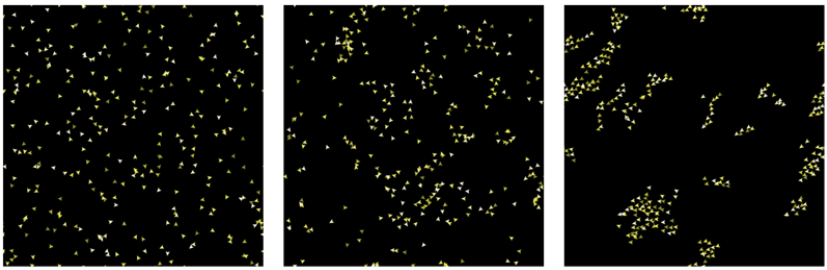

Figure 1. Screenshots of the flocking model where agents form flocks using simple local movement rules.

Play with the model and see what happens if you change the vision, the minimum-separation threshold, and the maximum adjustment degrees.

Reynolds animated flocks. There is still a lot of research on how birds and fish actually behave collectively. Nowadays, we can make more detailed measurements of those groups to test those models. For more information see http://collectivebehaviour.com/ and here.

Heroes and Cowards

To demonstrate the consequence of small rule changes, you can do a number of experiments with a group. I do this sometimes with my class at the beginning of a modeling course. It works well if you have more than 20 participants and do this in an open space. Each individual randomly picks out two other participants. Those others are called A and B. In the first experiment, you expect to move your position in the open space so that B is between you and A. Think

about A being an attacker, and B is the only person who can protect you. Everyone has their own A and B. What will happen? You may try it out with some friends, but if you don't try to imagine what might happen. In the second experiment, you will take the position of B and keep yourself position between A and B. Will this make a difference?

The origins of this activity are not precisely documented, but it has been around since the late 1990s (Bonabeau, 2002). In NetLogo, you can find a simulation of these experiments (see also NetLogo Web version here). If you do the exercise with a group, you will realize that the precise outcome is unpredictable, but there is an emergent pattern. The model demonstrates those emergent patterns for the 2 experiments, and some interesting variations. The model Heroes and Cowards can be found in NetLogo library IABM Textbook/chapter 2.

For the first experiment, we select "cowardly" for the Chooser "personalities." Hence the agent adjusts its position to keep B between itself and A. The agent will face away from the friend and enemy trying to keep the friend between itself and the enemy (Figure 2).

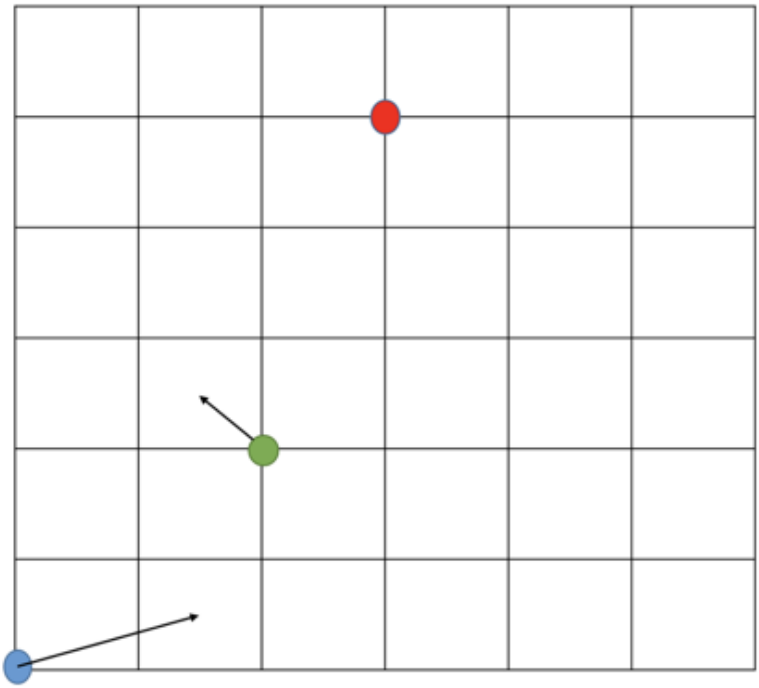

Figure 2. In this Figure, we see the hero (green), coward (blue), and enemy (red) agents. If the agent is a coward, it will adjust the heading as indicated in the figure with the arrow from the blue agent, and if it is a hero, the heading will be adjusted as depicted by the arrow for the green agent.

```
facexy [xcor] of friend + ([xcor] of friend - [xcor] of ene
  fd 0.1
```

The result is that the agents all end up crawling up the wall of the simulated environment, walking away (Figure 3). If you take away the border by making the landscape a donut shape landscape instead of a square, you get agents randomly moving around.

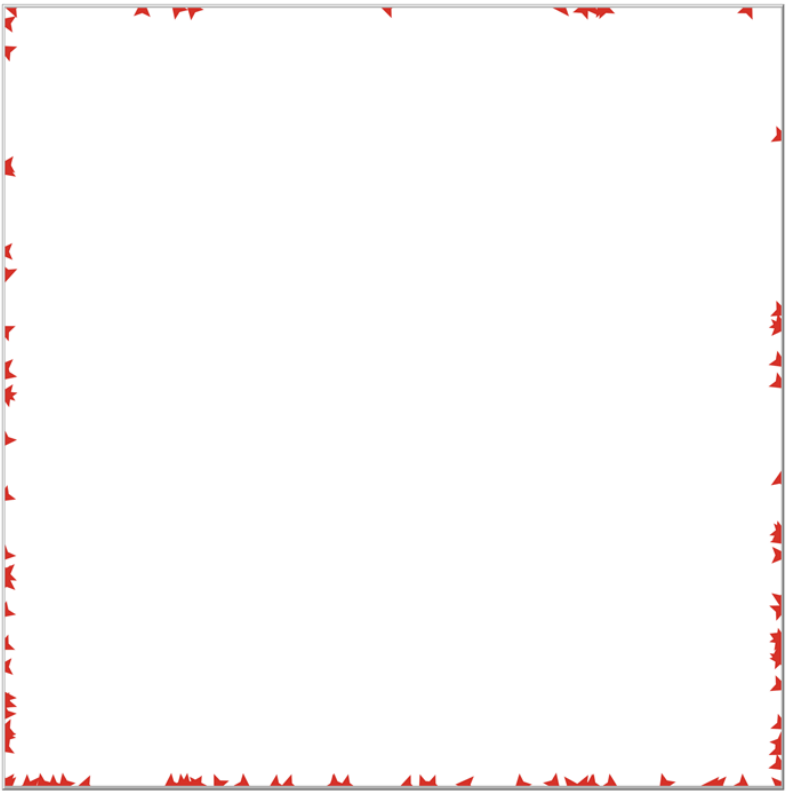

Figure 3. When all agents are cowards they end up positioning themselves so that their friend is between the enemy and self.

If we look at experiment 2, we select agents with brave personalities. In that case, the agents will move towards the middle position between friend and enemy. The result of this simulation is that all agents will end up in one point, all trying the stay between A and B. If you do this with an actual group of people, you may not get exactly this outcome. Besides, that people have some personal space and are not dots on a screen, they may also not randomly select others. Perhaps some individuals are selected more to be protected than others, which can lead to different behavior types.

```
facexy ([xcor] of friend + [xcor] of enemy) / 2
       ([ycor] of friend + [ycor] of enemy) / 2
fd 0.1
```

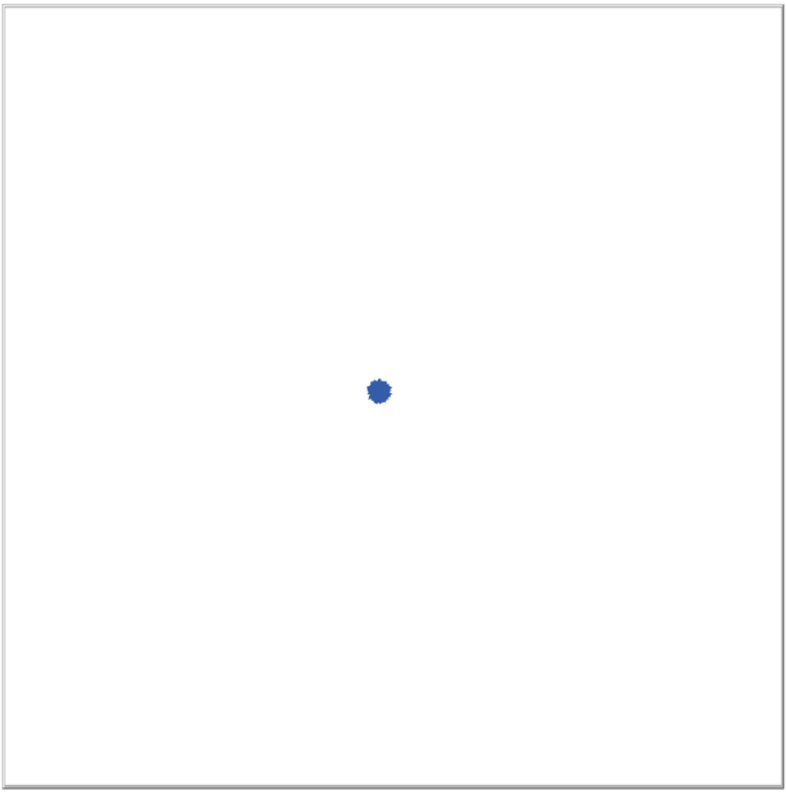

Figure 4. When all agents are brave they end up competing to be between the enemy and friend in the center of the space.

Now we have a model of this exercise; we can explore variations of the game. What if half of the group play according to experiment 1 and the other half according to experiment 2. We will see that quite interesting dynamics emerging (Figure 5). In fact, some people have explored many different random seeds, and you will see a selection of remarkable outcomes when you click on buttons in the set of

preset configurations. If you select a button and click on edit, you see that clicking on the button activates the procedure preset with a particular value of the seed. The very different outcomes for the various buttons is caused by different initial positions of the agents and their selection of enemies and friend. All the runs have the same rules with the same number of agents. As such, this model demonstrates that changing a rule can lead to a very different outcome, but even if you don't change the rules, but just the initial positions of the agents, very different outcomes can be observed.

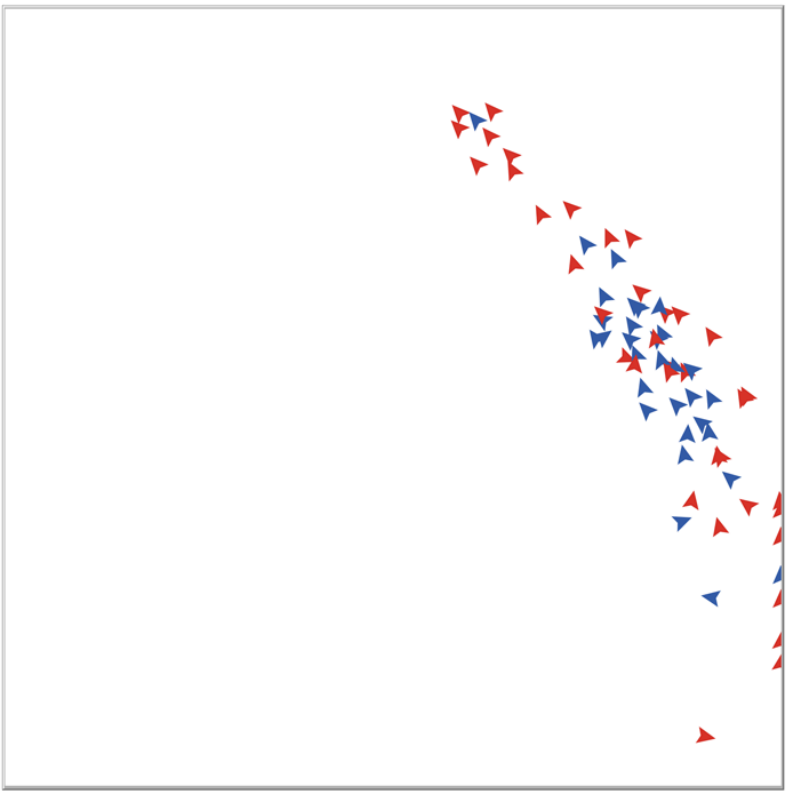

Figure 5. Example of results with mixed populations of cowards (red) and heroes (blue).

Creating Paths

You may have taken a short cut through the grass on your route to home yesterday. Where did you walk? You likely walked in the footsteps of others who took a short cut before you. In fact, we can simulate the creation of short cuts using the model Path in the Social Science Library of NetLogo (See also NetLogo Web version here). Agents have goals where to go to. If there is not a building on the map, they select a random patch in the landscape. Each tick the agent selects a patch to move to that is closer to the goal, but it only considers patches that others have trembled on recently. When patches are not used, they recover, and people may not select them since they do not notice others have walked on it in the past. When we simulate this for the situation where agents pick random patches as targets, we see the emergence of regular paths (Figure 6). If we add some houses that function as targets for the agents, we can create new sets of routes (Figure 7).

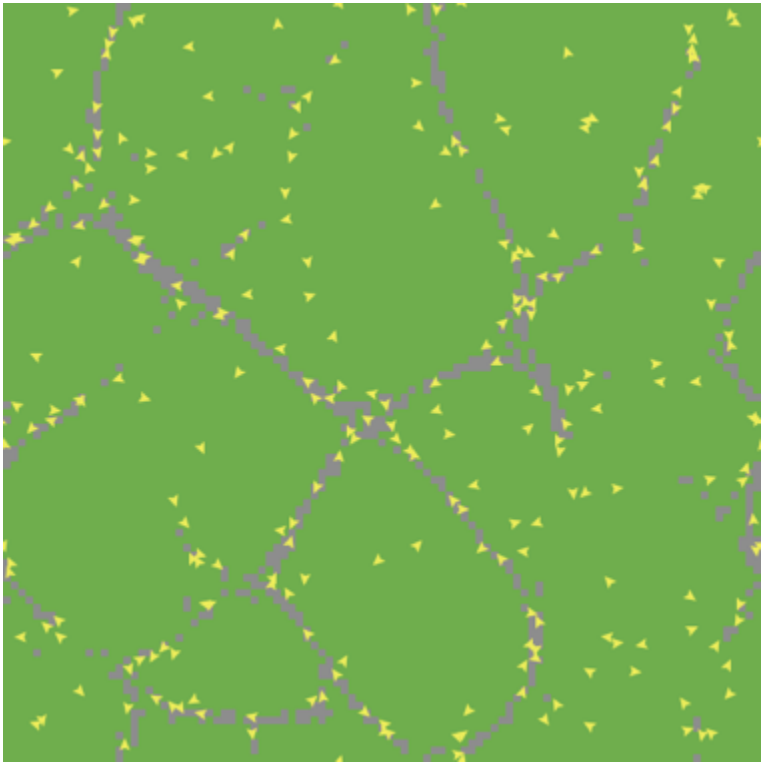

Figure 6. Emergence of regular paths.

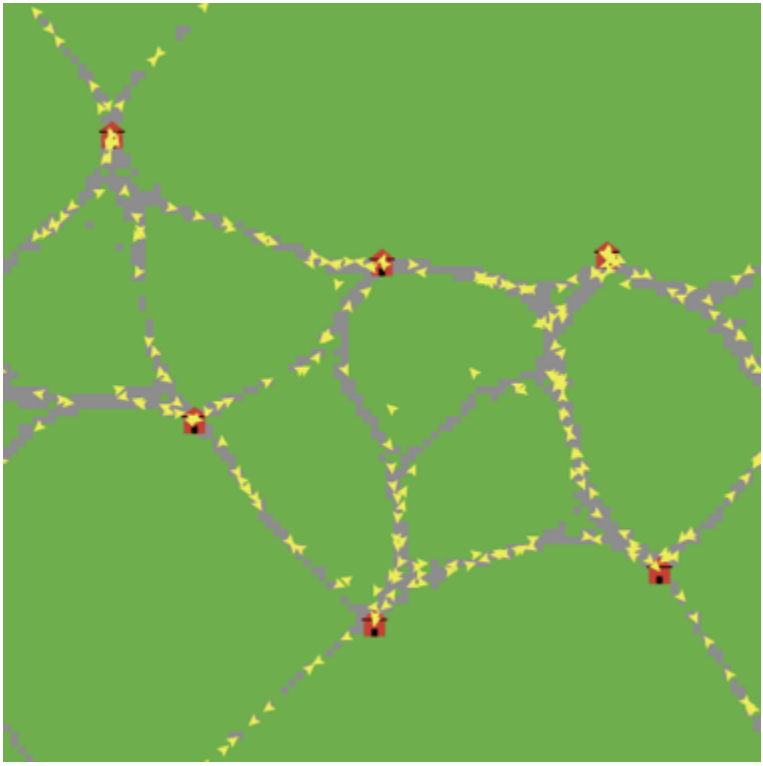

Figure 7. Including houses as goals for agents.

By adjusting how fast the popularity of a patch as a possible patch for a path is declining, we can see different types of paths emerging. If past use of patches is rapidly declining, we do not see paths emerging. In fact, no clear marks are left in the landscape leading agents to walk over the grass.

Ants Foraging for Food

When ants find food and bring it to their nest, they leave pheromones in the environment that other ants can use as cues to where to go too. In the model Ants in the Biology folder of NetLogo library, you find a model that demonstrates how leaving pheromones to facilitate coordination among ants (for NetLogo Web version click here).

Given are 3 food sources, and when we run the model, we see that the food sources closest to the nest are harvested first. The reason is that ants, who without pheromone cues randomly move on the landscape, are more likely to bump into the closest food source. If they do, they leave a trail of pheromones rallying other ants to move to the food source, until the food source is emptied and the pheromone trail evaporates over time.

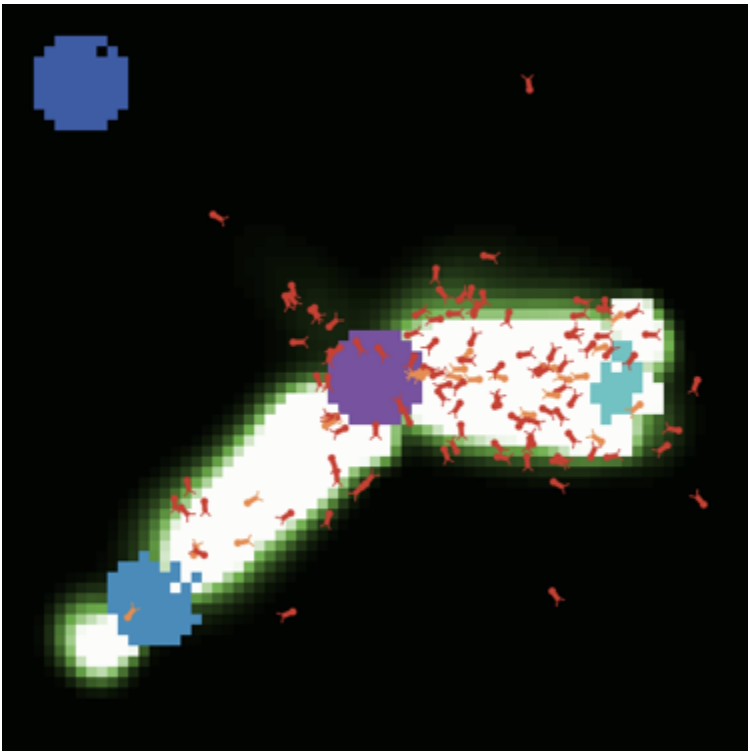

Figure 8. Screenshot of the simulation. The purple circle in the center is an ant nest, the other three circles represent food resources.

Food sources that are further away have difficulty getting a strong pheromone trail since it takes longer for the ants to walk that distance (and more pheromones are evaporated). We can vary the level of evaporation, and we could expect that the evaporation rate

will affect the dynamics of the model. In Figure 9, we show the meantime for ant colonies to harvest food sources 1, 2, and 3, where 1 is the closest, and 3 is the one further away. If there is no evaporation, agents will not get a good signal since the whole environment will become saturated with pheromones. Hence having a certain level of evaporation increases the quality of the pheromone signal. As Figure 9 shows, this leads to a faster depletion of the resources. But if the evaporation rate increases more, the signal from far-away-resources disappear too quickly for agents to discover. The focus remains on bringing agents to the closest resource. If the evaporation rate becomes really large, there is no real information signal anymore, and the ant colony starts to perform as if there is no pheromone signal. To conclude, there is an optimal evaporation rate, around 5% per tick.

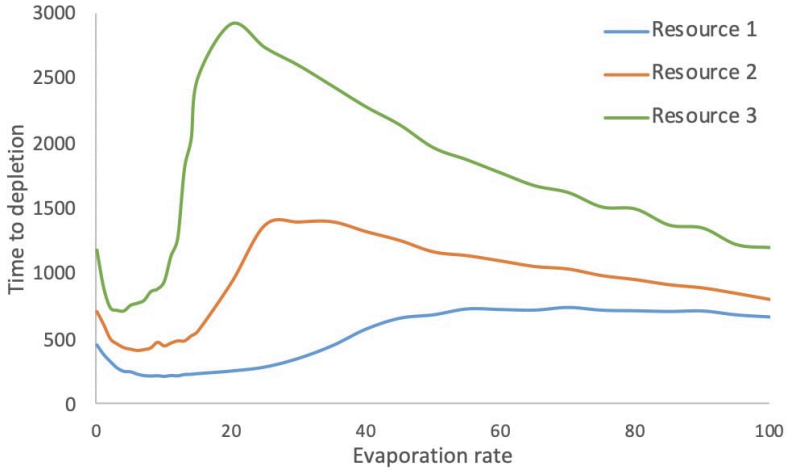

Figure 9. Impact of evaporation rate on the speed in which resources are consumed.

Summary

This chapter discussed some classic models of agents forming spatial patterns using simple rules that take into account the

position of the agent itself, the position of nearby agents, and the conditions of the local environment. In Heroes and Cowards, we see that a mix of different types of rules within the agent population can lead to very different behavior compared to the situation that all agents have the same rule. Agents can communicate by changing the environment by dropping pheromones or tremble on the landscape.

In all, the models here still consider agents reactive to local cues leading to macro-level patterns.

References

Bonabeau, Eric (2002) Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences of the United States of America* 99: 7280-7287.

Bonabeau, Eric, Marco Dorigo, and Guy Theraulaz (1999) Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, New York, NY.

Deneubourg, Jean-Louis, Serge Aron, Simon Goss, and Jacques M. Pasteels (1990). The self-organizing exploratory pattern of the argentine ant. *Insect Behavior* 3:159–168.

Helbing, Dirk, Joachim Keltsch, Péter Molnár (1997) Modelling the evolution of human trail systems. *Nature* 388: 47–50.

Reynolds, Craig (1987) Flocks, herds, and schools: A distributed behavioral model. SIGGRAPH ’87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques. Association for Computing Machinery: 25–34

CHAPTER 9

Sugarscape

Key Concepts

In this chapter, we will:

- introduce the classic Sugarscape model of Axtell and Epstein,
- explore some basic configurations of the Sugarscape model,
- demonstrate how inequality emergences from simple rules,
- include spice to the model and allow agents to trade.

In 1996 Joshua Epstein and Robert Axtell published their classic book Growing Artificial Societies in which they presented their model Sugarscape. The model was inspired by the work of Schelling (Micromotives and Macrobehavior) and Conway (Game of Life), which we discussed in previous chapters. Sugarscape provides a concise exploration of the possibilities of agent-based modeling for the social sciences. It demonstrates the possibility of integrating insights from demographics, ecology, economics, sociology, and political science into one model and using simple rules to grow

macro-level patterns that are observed in reality. However, the model is an artificial society and makes many simplistic assumptions. Nevertheless, it has inspired the ABM community to build those kinds of artificial societies to do social science. This chapter will explore a number of the basic runs of the Sugarscape model using models in the NetLogo library (Social Science / Economics / Sugarscape).

Resource Growth and Extraction

Sugarscape consists of agents who live on a torus, a donut-shaped landscape. In this landscape, there are resources, sugar. Each patch has a maximum level of sugar, a carrying capacity, and the value of the maximum level is spatially diverse (Figure 1). The maximum sugar level is highest in the left bottom and top right corner of the landscape.

Figure 1. The maximum levels of sugar on the torus landscape. The more intense the yellow, the higher the maximum level.

Agents need food, the sugar, to survive. Each tick they burn a certain amount of sugar, according to their metabolism rate. If the sugar level drops to zero, because it does not derive enough food to compensate for the metabolism, the agent will die. Agents also have a vision, which is the number of patches it can see around the current patch, to the left, right, bottom, and top of the agent. Initially, agents are randomly distributed on the landscape, and they have different levels of vision and metabolism, as well as a different level of the initial amount of sugar.

We will now discuss how the agent decides where to move to.

Each agent has a vision, and since agents can only look in straight lines, we define a set of patches that the agent can see. This is done by specifying the attribute vision-points. Using the primitive range, a list of 1 to vision + 1 is created, and for each vision level, a list of lists is created in each of the four directions. Basically, vision-points are the patches to the four directions as far as the vision can reach.

```
set vision-points []
foreach (range 1 (vision + 1)) [ n ->
  set vision-points sentence vision-points (list (list 0
]
```

In the move procedure, the agent defines a set of patches composed of the current patch and the patches in her vision, which are not occupied. Those are the patches considered to be of interest since only one agent can occupy a patch. From those candidate patches, the agent calculates the maximum level of sugar available on the available patches. Suppose there are possible empty patches with sugar; the agent moves to the closest of those patches which have the maximum level of sugar. Note that this implicitly assumes that agents can move a number of patches in one tick. In fact, agents with a higher vision can move more patches if needed.

```
let move-candidates (patch-set patch-here (patches at-poi
let possible-winners move-candidates with-max [psugar]
if any? possible-winners [
  move-to min-one-of possible-winners [distance myself]
]
```

The first simulation we will look at is the Sugarscape 1 Immediate Growback which can be found in the folder Social Science/Economics/Sugarscape. In this model, the resource grows immediately back to the maximum sugar level after the agent has consumed the resource during that tick. What will happen when we start with agents randomly scattered on the landscape? Some agents will die. They are initially in an area with a low maximum

sugar level and do not have enough vision to move to a location that can support their metabolism rate. It is more likely that agents with a high metabolism rate and/or a low vision die and thus the average metabolism rate declines and the average vision increases. After a few ticks, the agents do not move anymore, since they are in a location that is best within their vision (Figure 2).

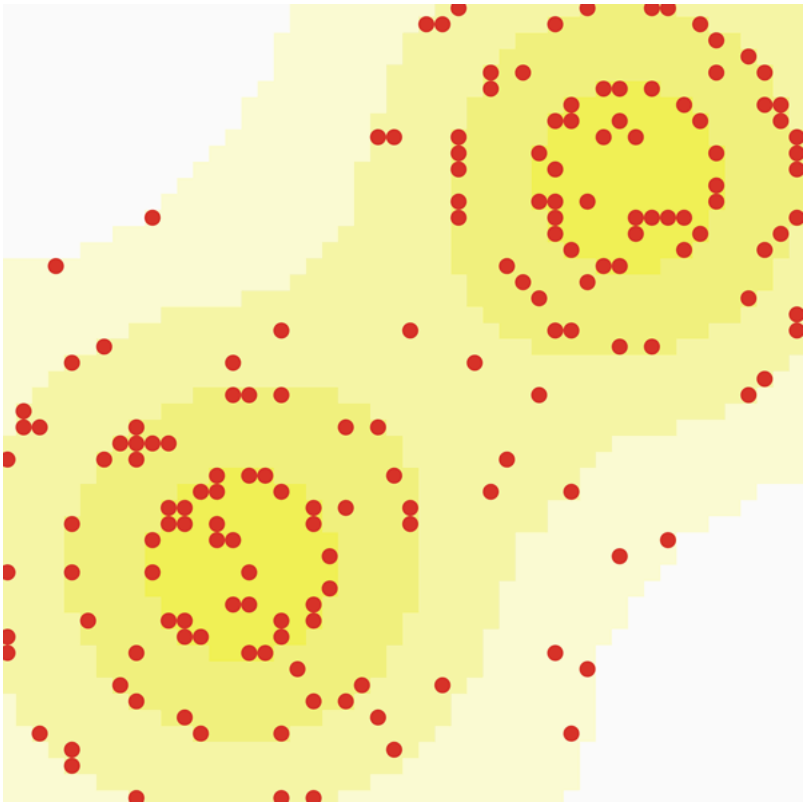

Figure 2. Screenshot of simulation when resource grows back immediately after harvesting.

Now suppose that the resource does not grow back immediately, but slowly with one unit per tick (Sugarscape 2 Constant Growback). If an agent consumes the whole patch, this patch is probably not

the best option for the next tick, and the agent may find another patch with a higher level of sugar. Now we see agents constantly moving. Still, the metabolism rate declines, and the vision rates increase due to the death of agents with a high metabolism rate and low vision (Figure 3).

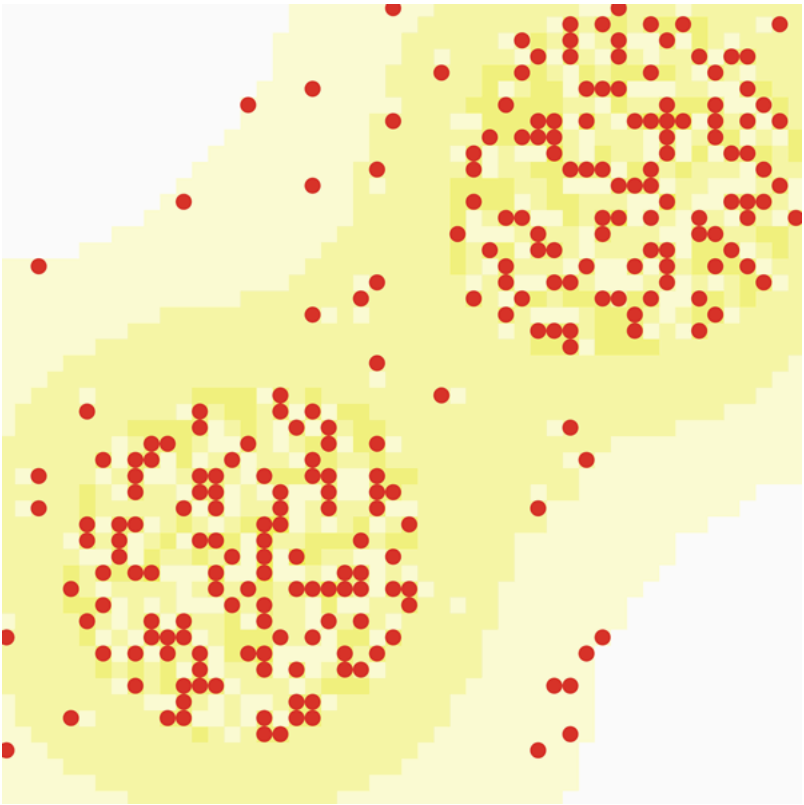

Figure 3. Screenshot of simulation when resource grows back one unit at the time.

In the third version of Sugarscape in the NetLogo library Sugarscape 3 Wealth Distribution, we make a few adjustments to the population dynamics of the agents (Figure 4). If an agent dies, a new agent with random values of the attributes is placed on

a random empty patch on the landscape. A new attribute of the agent is her age. Agents also have a maximum age, and if the maximum age is passed, the agent will die, even if it still has sugar.

When we run the model, we see in the plot Wealth Distribution that most agents have a small amount of sugar, and a few have a large amount. Even though agents do not inherit sugar resources from their parents, we will get wealth inequality (Figure 5).

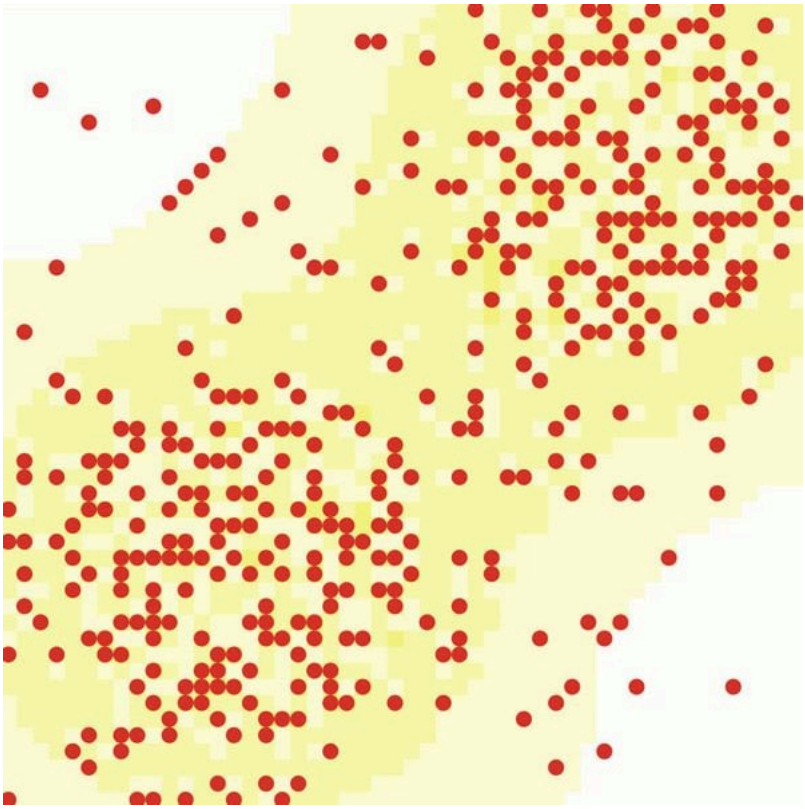

Figure 4. Screenshot of simulation when the agent population has a fixed population size and maximum lifetime.

A more formal way to measure this is the Gini coefficient. To calculate the Gini coefficient, we imagine putting all agents in a row

from the poorest to the richest. This will generate the Lorenz curve. If each agent has the same wealth, we would see a straight line from zero to the total amount of wealth in the population. If there is inequality, the resulting line of accumulated wealth is below the straight line. The Gini coefficient is the difference between the red and black line (surface) divided by the surface below the black line. Hence if everyone is equal, the Gini coefficient is zero. Suppose there is a high level of inequality; the Gini coefficient moves above 0.5. In terms of income inequality, the Gini coefficient of major economies is between 0.25 and 0.4 after taxes are paid.

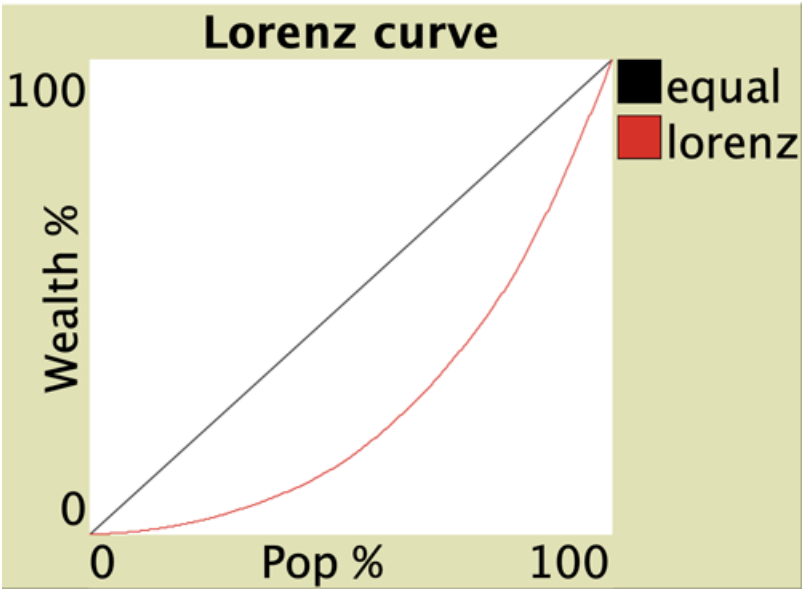

Figure 5. The Lorenz curve for a certain tick during a sample simulation of Sugarscape.

Sugar and Spice

In this section, we will add some spice to the model. The NetLogo model can be downloaded here, but we urge you to implement the model using the basic Sugarscape model from the previous section. Agents are assumed not only to need sugar but also spice.

There will be two resources, sugar and spice, and each agent has a metabolism for sugar as well as for spice. Some agents will need more sugar to survive and others more spice.

By needing two resources, which are spatially located in different areas of the map (Figure 6), we create opportunities to trade. Agents can trade with each other at a local level if they both think it will make them better off. We will demonstrate that this local trading, where prices are determined locally by bargaining, leads to an average price of sugar and spice that is the same as the theoretical optimum when agents have unlimited knowledge.

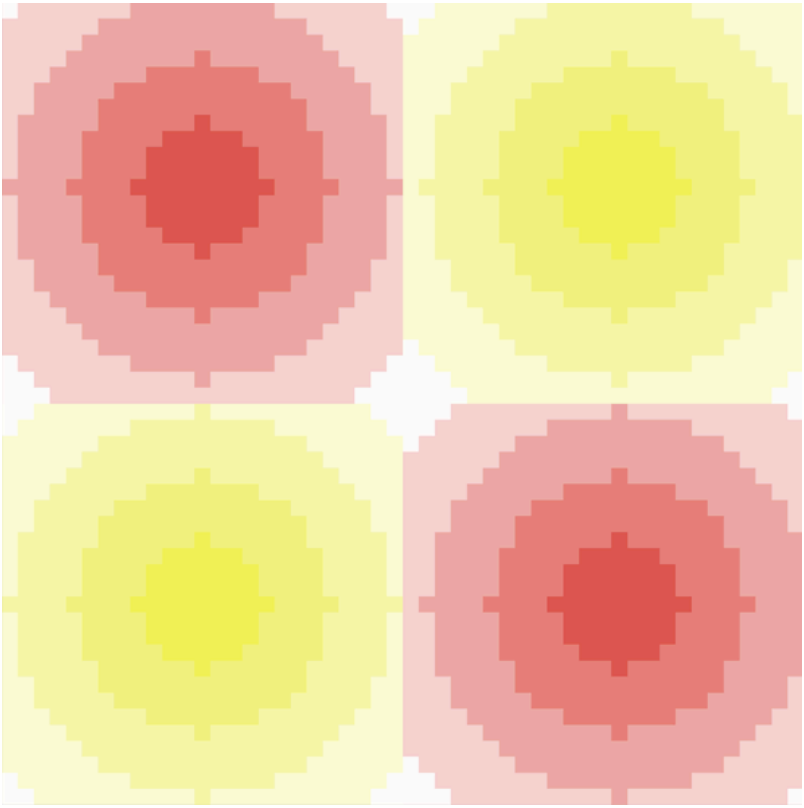

Figure 6. Resource landscape of sugar (yellow) and spice (red).

How do agents decide whether they want sugar or spice? Since an agent will die if it runs out of sugar or spice, the expected duration of the current accumulated resources for sugar (w_1) and spice (w_2), given the metabolism rates for sugar (m_1) and spice (m_2), gives an indication which resource is more desirable. The expected durations are w_1 / m_1 for sugar and w_2 / m_2 for spice.

The welfare function, W, for the agents is now defined as a Cobb-Douglas function, by

$$W(w_1, w_2) = w_1^{m1/(m1+m2)} w_2^{m2/(m1+m2)}$$

When agents decide where to move, they will make decisions based on the expected wealth after agents collected the sugar or spice from that patch. Hence decisions are not based on the amount of sugar on a patch as in the previous version.

We include in this model the reproduction of agents. This is done in a very simplistic manner. If an agent has wealth beyond a certain level, it will generate an offspring if there is an empty neighboring patch. The parent will half the levels of sugar and spice, and the offspring will derive the other half of the resources. The attributes of the parent, metabolism rates, and vision, are copied to the offspring with a small probability of errors that adjust the attributes' values.

We will now have endogenous population levels. A typical simulation shows an initial increase in the population size, which reduces the sugar and spice resource levels, followed by a decline of the population levels (Figure 7). A group of agents lives on the border of spice/sugar resources, where there is a little bit of both. Others who need more sugar and or spice can only survive if they have a high vision and thus move back and forth between the sugar and spice resources (Figure 8). Figure 9 shows that we see that mainly agents survive who have a low metabolism, so they have

enough time to move between sugar and spice to get enough of both. The average vision also improves a bit, since having a better vision allow agents to have a larger movement radius.

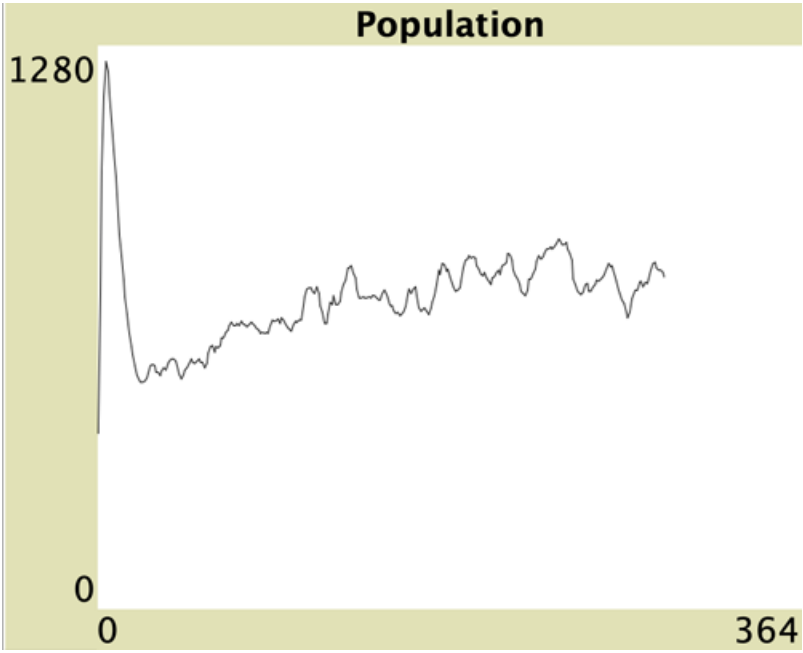

Figure 7. Population-level of a sample simulation.

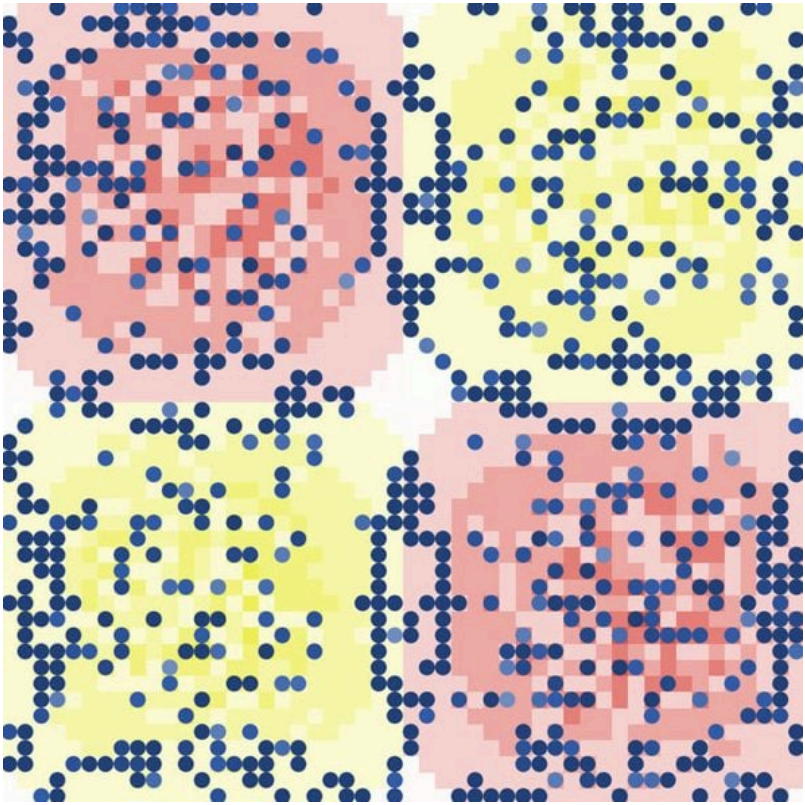

Figure 8. Screenshot of the simulation, where blue dots represent the location of the agents.

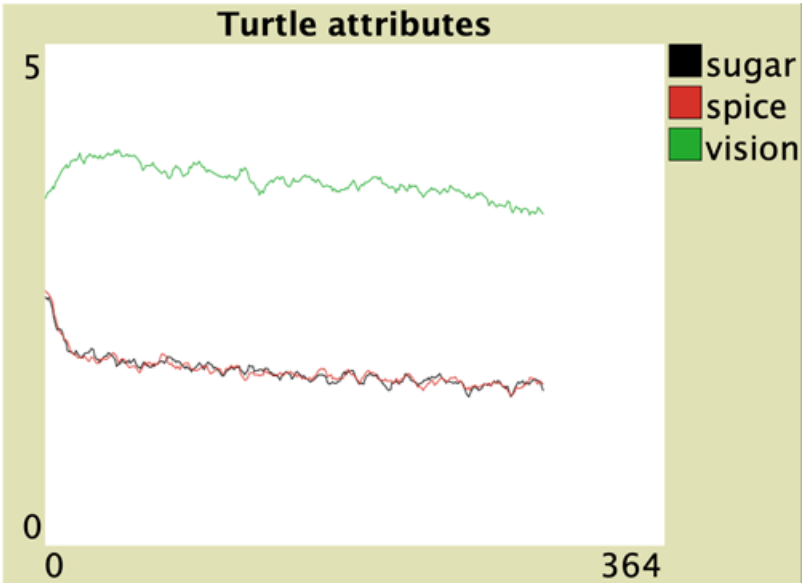

Figure 9. Average levels of metabolism rates of sugar and spice of the agents and the average vision level of the agents.

What if agents are allowed to trade their surpluses? An agent who burns more sugar may come across an agent who needs more spice, and they may exchange some of their resources. What would those exchanges look like? What will the price be that is used in those individual encounters? And will trade improve the wealth of the agents?

Chapter 4 of Epstein and Axtell (1996) discusses the details on how to derive the trade rules. Here we just discuss the rules derived. Using the accumulated resource by an agent, and the metabolism rate, the relative scarcity of a resource can be calculated. Agents can calculate the marginal rate of substitution (MRS) of agents in the neighborhood:

$MRS = (w_2 / m_2) / (w_1 / m_1)$

If MRS of agent A is bigger than the MRS of agent B, A buys sugar

from B who gives spice in return. The reason for this is that B has a relative surplus of spice compared to agent A.

The trading price that the agents will set on is $p = \sqrt{MRS_A * MRS_B}$. This means that if $p > 1$, p units of spice are exchanged for 1 unit of sugar, and if $p < 1$, then 1 unit of spice is exchanged for 1/p of sugar. Before the trade is made, both agents will be checked to benefit from the trade in their wealth level.

We see that agents have many trade events (Figure 10). The increase in trade events relates to the evolution of agent types. A reduction of the mean metabolism rates, allows agents to have surpluses to trade. We also see that the mean price for sugar is close to 1 (Figure 11), the expected outcome if there was a general equilibrium with the same amount of spice and sugar, and the preferences (metabolism rates) similar among the population.

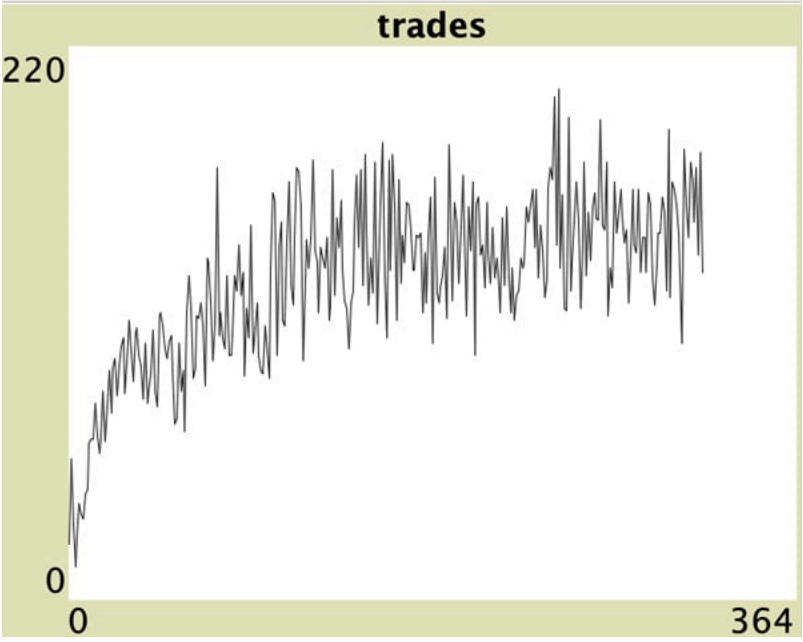

Figure 10. The number of trades per tick overtime.

Figure 11. The average price of sugar per tick of all trades during that tick.

The spatial pattern of the agents on the landscape seems similar to the case when there was no trade (Figure 12 vs 8). However, agents do not all have to jump between spice and sugar, since they can trade with their neighbors to keep their resources replenished.

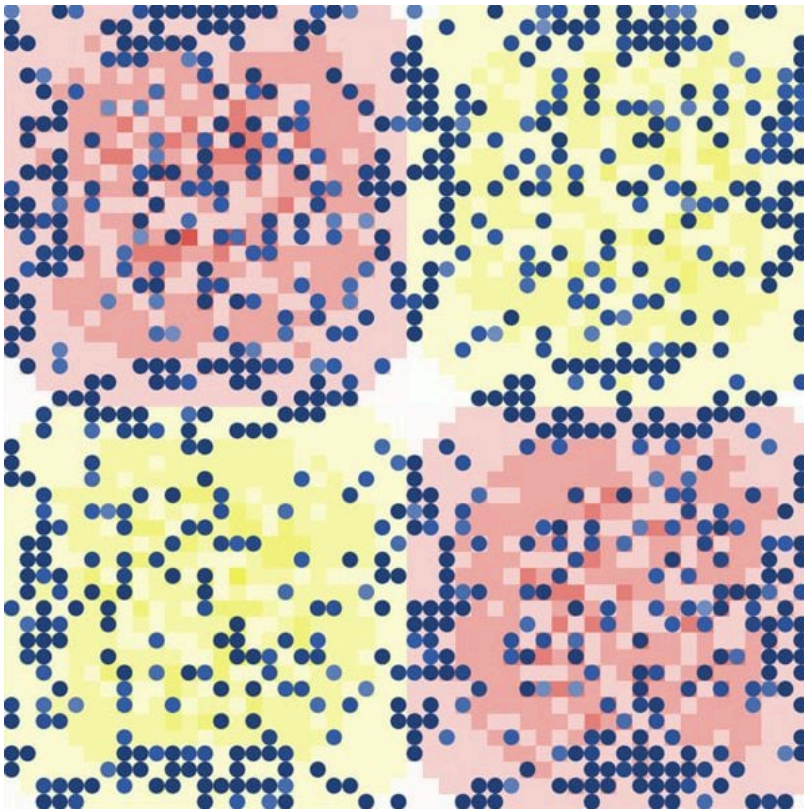

Figure 12. Screenshot of agents of the resource landscape when they can trade.

One consequence of the inclusion of trade is that the carrying capacity of the landscape is increasing. In Figure 13, you see the results of the population size after 250 ticks for 1000 runs for each parameter setting. The results depicted are for the conditions where agents are allowed to trade (true) or not (false) and different thresholds of wealth at which agents reproduce. If the reproduction threshold is higher, there will be less reproduction possible, and the carrying capacity is lower.

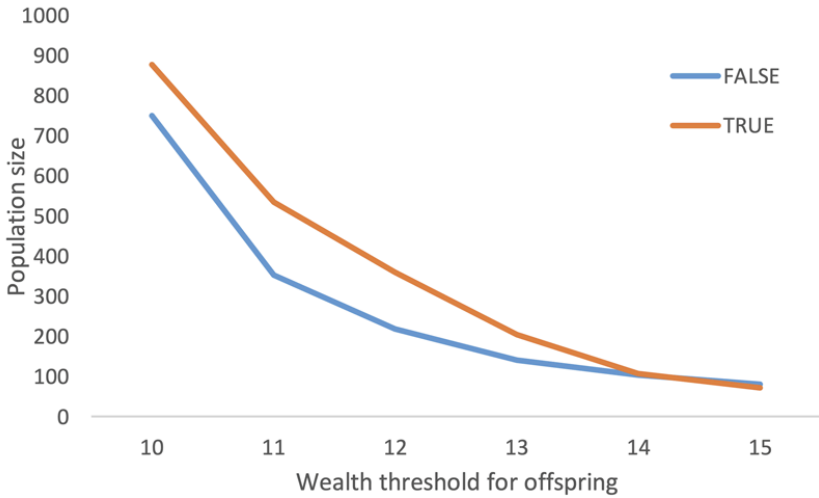

Figure 13. Average population levels for different reproduction thresholds and including trade or not.

There are many possibilities to extend this model. The Sugarscape model is a classic in the field as it demonstrates the wealth of opportunities for computer simulation of artificial societies. Some adjustments that you may want to explore with the model described in this section.

– Allow agents only to move one patch a tick.

– Allow agents to have offspring if there is an empty patch within their vision.

– Allow agents only to harvest sugar or spice from a patch.

– Explore different reproduction thresholds, such as accumulated amounts of sugar and spice independent of metabolism rates.

– Allow for different orders of updating, such as wealthy agents make first choices.

Let me know what you find.

Summary

Sugarscape is a classic model among agent-based models since it demonstrated the potential to integrate many different topics

within one model. The model includes more strategic reasoning, such as agent bargaining over sugar and spice. As such, the agents move beyond simple reactive agents.

References

Epstein, Joshua M., and Robert Axtell (1996) Growing Artificial Societies: Social Sciences from Bottom Up. Brookings Institution Press and The MIT Press.

Video of the original book summary

PART III

MODEL ANALYSIS

This part of the book dives into procedures and practices of proper model analysis. Instead of looking at each simulation on your screen, there are powerful tools available in the NetLogo ecosystem that allows for systematic exploration of the model. We discuss BehaviorSpace and BehaviorSearch and how to use them?

CHAPTER 10

Doing Research with Models

Key Concepts

In this chapter, we will:

- discuss the kind of research questions you can address with agent-based models,
- present the modeling cycle,
- provide practical tips on how to pursue the modeling process,
- introduce you to the ODD model documentation protocol.

In this textbook, we discuss agent-based modeling as a tool for doing research. In this chapter, we discuss some practical aspects of the art of modeling.

When we talk about using models for research, we typically want to use a model to derive more insights about a specific research question. Defining a good research question is one of the hardest aspects of the modeling cycle (Figure 1). A common way is to start with theory and define what question could be addressed with an agent-based model. A question like "why do not more people adopt an electric vehicle?" is difficult to answer with an agent-

based model since it is an empirical question for which you need to do survey research or interviews with potential buyers of cars. However, we could address a question like "Based on our understanding of consumer behavior and historical trends of car sales, what could be potential trends of electric vehicle sales in the coming years?". This question could be addressed by a variety of modeling approaches, including statistical models. A question that focuses more on the agent-based modeling methodology would focus on social influence and social network structure, assuming you would have relevant data for those components.

What are strategies that could help you to define an appropriate research question? Read publications of modeling studies in your interest area and see what kind of questions they address. Good research questions using agent-based models focus on how mechanisms affect the behavior of agents and their environmental impact on the outcome of interest. This is not surprising since agent-based models are defined by those mechanisms, the algorithms that define the actions of the agents.

A common mistake is that one perceives a model as a kind of virtual reality and defines questions that can only be addressed by empirical observations of the actual systems. A model can provide the consequences of algorithmic statements, and you may derive an understanding of why certain behavioral mechanisms are more suitable for one topic compared to another. However, you cannot use an agent-based model to answer direct questions about why people you represent in the model have certain beliefs and motivations. The fact that we model agents with a particular algorithmic statement is a major simplification of how people actually make decisions, and why people make certain decisions is not something we can answer directly. However, a researcher may derive a better understanding of a system and get a good idea of what empirical research is needed to answer an empirical question on the motivations of decision making.

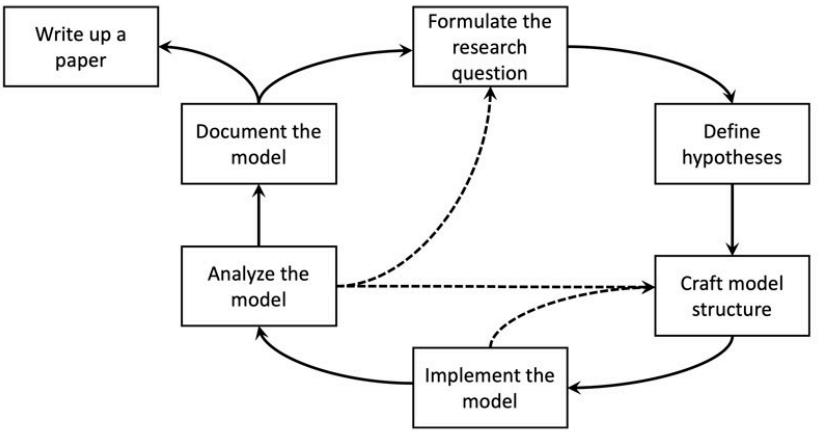

Figure 1. Modeling cycle. Based on Figure 1.3 of Railsback and Grimm (2012).

Once we have defined a research question, we can define hypotheses, which will help formulate what outputs to investigate and what mechanisms to include in a model. For example, "is racial discrimination the sole reason for segregation?". To test this, we could only derive segregation in a model where agents are only "happy" if the majority of their neighbors are the same as them. The model of Schelling demonstrates that segregation also happens if agents are fine in living in neighborhoods where the majority is different from them.

Now we have a better idea of the questions we like to pose; we start crafting a model. Also, here it is helpful to look at publications of models like the one you want to implement. In fact, I would suggest you replicate one or more models related to your topic of interest, so you get a better idea about what kind of models are out there, and the challenges others have experienced. It is not unusual for modelers to be overly optimistic in their ambitions. Therefore getting a reality check on the details of an existing model will help derive a more realistic scope for the modeling activities.

An important guideline in crafting a model is to keep the model as simple as possible, but no simpler than that. This is easier said

than done. In fact, simple models are difficult and time consuming to craft, while it is easy to create complicated models. Simple models, that capture the essence of the topic of interest are the results of many iterations of tinkering with the model. As such, modeling is an art, like creating a sculpture or painting, where the ultimate sculpture or painting consists of just a few "brush strokes" to visualize the essence.

Use flow charts to draw the connections between the components that you want to model. What are the feedback processes? What are the temporal and spatial scales? What are the kind of agents you include and their basic decisions and interactions?

Knowing that modeling is a process, don't spend all-time on the drawing board. Having done replication of a related model, or building on a previous study helps you not start from scratch. Implement your first ideas, and you may find out that your ideas might be less brilliant as assumed. The modeling process, the programming, and seeing the results of assumptions you made, is a learning process that helps you to redefine the model structure.

Once you have an initial version of the model, do a proper analysis. In the end, you may spend more time on model analysis compared with coding the model. Explore extreme conditions of the model and test whether the model will still produce relevant outcomes. For example, will you get ants forming trails if the pheromones evaporate immediately? If the population increases, will the resource dynamics follow expected trends? What if the population levels increase twice as fast?

When you start having confidence in the workings of the model, you can start developing a more formal analysis. A basic analysis will be a sensitivity analysis where you vary parameters systematically. You can start by varying the parameters one by one to identify which parameters impact the model outcomes most? Subsequently, you can systematically vary two parameters or do a

comprehensive variance-based sensitivity analysis. We will discuss some examples in the coming chapter.

Such a sensitivity analysis will help you understand the model better and may lead to adjustments in the model or even reconsider the research question you are addressing. Other types of analysis could be done related to the research questions you have and whether you have qualitative or quantitative data to evaluate your model.

Qualitative data can be used to inform the assumptions about the model mechanisms. Focus group discussions could reveal certain practices of how households make their choices on how to cope with resource scarcity. You may derive information about the kind of adaptation strategies available and can use the model to explore how well such adaptation strategies cope with distributions of resource availability. Even with qualitative data, you can make a quantitative model. In fact, many insights we have derived from the example models in previous chapters are qualitative, namely how pheromones impact movement of ants, how paths are formed, how segregation and flocking patterns could emerge, etc.

If you have quantitative data, you can use this to define directly parameter values from observations, such as the observed amount of kilograms of fish caught when fishing recreationally (you may use the mean or draw from an empirically informed probability distribution). Quantitative data can also be used to evaluate your model. How do the model outcomes match with the empirical observations? It is actually not evident what we mean with a "match." Suppose you have a model that generates the water use of households, do we mean with a good match a close approximation of the simulated and actual mean water use per day, or do we want to use the temporal patterns during the day, weeks or months, or take into account the household types? If we include more information to evaluate the match, do all kinds of information weigh the same, or do some kinds of information weigh more?

In a later chapter, we will present some examples of calibrating

a model to explain empirical observations and show some of the challenges in evaluating models with empirical data. Besides formal quantitative tests, there are also subjective processes such as deriving feedback from relevant stakeholders or even doing a Turing test with stakeholders who are asked to choose between simulated and real data. In the end, there is no one objective way to define whether your model is a good representation of the object of your study. By doing systematic evaluations and deriving feedback from different experts, including stakeholders, you may obtain more confidence, but it remains a subjective evaluation.

When you do a systematic model analysis, using data or not, be critical about your model. I am always hesitant to trust the outcomes of the model and need to fully understand why I get certain outcomes if they are non-intuitive. Since the model is just a set of logical statements, you can trace back why you get the outcomes you get. Yes, you may be surprised by some model results, but in the end, you need to understand why and the model outcomes should not be surprising at all.

During the process of implementing the model, it is a good practice to start documenting your model. Good documentation of an agent-based model is not easy since those models are typically beyond a set of equations. A protocol that can help you to develop good systematic documentation is the ODD protocol, described in detail in Grimm et al. (2010, 2020). ODD stands for "Overview, Design Concepts, and Details. The overview provides information for what the model is about, then a description of various concepts behind the model, and finally, a discussion of details like modules of the model (Figure 2).

The Overview starts with defining the Purpose of the model. What is the research question that is addressed by the model? Knowing the research question will help the reader understand why some design decisions are made, such as simplifications of processes that might be important for other research questions.

Next, we discuss the components of the model, such as the types

of agents and patches you include in the model, as well as the important attributes of those components. What are the temporal and spatial scales of the model? What are the global variables of the model, if any?

When we describe the process overview and schedule, it will be helpful to craft a flow diagram about how the various components and processes of those components interact with each other. What are the actions the model takes during each tick?

The Design concepts describe a number of basic concepts that will help the reader how the model is implemented. Do agents learn, predict the actions of others, are members of groups? Does the model include stochastic processes and evolution?

The details of the model start with a description of the initial conditions of the model. What are the initial values of the attributes of the agents, how are they allocated on the landscape, what is the initial memory or storage?

If you have input data, what is this data, and how is this data derived?

Finally, you describe submodels, like procedures in Netlogo. This could be done using mathematical equations of pseudocode.

Elements of the ODD protocol	
Overview	1. Purpose 2. Entities, state variables, and scales 3. Process overview and scheduling
Design concepts	1. Design concepts: • Basic principles • Emergence • Adaptation • Objectives • Learning • Prediction • Sensing • Interaction • Stochasticity • Collectives • Observation
Details	1. Initialization 2. Input Data 3. Submodels

Figure 2. Overview of the ODD protocol for describing ABMs.

The ODD protocol is guidance in documenting a model. Using the ODD protocol is no guarantee that you have good documentation. Often I notice that modelers using the ODD protocol describe their models in words and neglect the use of mathematical equations, pseudocode, and flow diagrams. Those last 3 items are critical in describing your model's structure and details, so keep them included.

When you have documented your model, you can archive your model code and documentation. A recommended model archive is the Computational Model Library of CoMSES Net (https://www.comses.net/codebases/). Archiving your work helps others build on your work, and make sure your future self also has the details of what you have done.

Finally, what did you learn about the modeling process? You may think that you do not need the model anymore now you have a much better understanding of the problem you were modeling. In writing up a manuscript about the model, emphasize the research questions, and focus on the model analysis in addressing the research questions. You do not have to include all details in a manuscript and move a lot of the technical details in an appendix.

Summary

In this chapter, we discussed the art of modeling and some practices that may help use modeling to address your research questions. Replication of existing models is useful to derive a good understanding of research related to your interests and improve your NetLogo programming skills.

References

Grimm, Volker, Uta Berger, Donald L. DeAngelis, J.Gary Polhill, Jarl Giske, and Steven F. Railsback (2010), The ODD protocol: A review and first update, Ecological Modeling 221: 2760-2768.

Grimm, Volker, Steven F. Railsback, Christian E. Vincenot, Ute Berger, Cara Gallagher, Donald L. DeAngelis, Bruce Edmonds, Jiaqi Ge, Jarl Giske, Jürgen Groeneveld, Alice S.A. Johnston, A. Milles, Jacob Nabe-Nielsen, J. Garry Polhill, Viktoriia Radchuk, Marie-Sophie Rohwäder, Richard A. Stillman, Jan Thiele and Daniel Ayllón (2020) The ODD Protocol for Describing Agent-Based and Other Simulation Models: A Second Update to Improve Clarity, Replication and Structural Realism, Journal of Artificial Societies and Social Simulation 23(2) 7.

Railsback, Steven F. and Volker Grimm (2012) Agent-Based and Individual-Based Modeling: A Practical Introduction, Princeton University Press

CHAPTER 11

Stochastic Processes

Key Concepts

In this chapter, we will:

- learn how to generate random numbers in NetLogo,
- Experience the difference between random updating of agents and updating in a fixed order.

In this chapter, we will look at various examples of randomness in NetLogo, how to implement randomness in NetLogo, and the consequences of different distributions. We look into this because agent-based models are largely stochastic nonlinear simulation models that require sophisticated systematic analysis. In the next chapters, we discuss tools for systematic analysis. In this chapter, we focus on the basics of the stochasticity in agent-based models.

Random Numbers in NetLogo

The basic command in NetLogo to generate random numbers is random. Random 100 means that an integer value is randomly drawn from the series 0, 1, 2, …97, 98, 99. If you type in show random 100 in the command center, then you will get a different value every time.

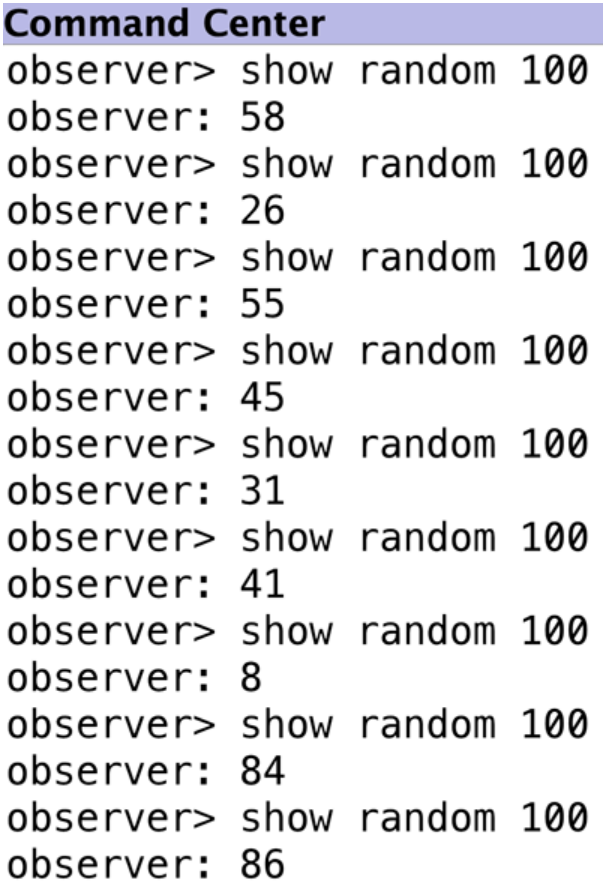

Figure 1. Screenshot of Command center after entering show random 100 various times.

If you want to simulate to flip a coin where 0 represents head and 1 tail, you will have to use random 2, which generates values 0 and 1. If you want to generate values from -100 to -1, you will have to subtract a value from the positive values generated by the random function like show random 100 – 100. Try it out!

As we discussed in Chapter 3 a random number is an outcome from a deterministic function. If you start the function with the

same number, you will get the same sequence. You can test this by first defining the value of the random-seed by say random-seed 10 in the command center. Then look at the numbers that appear if you use show random 100 a few times. Now type in random-seed 10 again and do again a few times show random 100. You will get the same sequence of random numbers. Basically, we had reset the sequence of "random" numbers when we put in random-seed 10 the second time. Also, here it is good to try it out and test that you may get a very different sequence if you use a different value for random-seed.

If you want to have real numbers instead of integer numbers, you have to use random-float. With random-float, the numbers will be between 0 and 100, more precisely in the interval [0,100).

Getting Lucky in Economic Systems

Can you become a millionaire by chance? Here we show a simple model suggesting that it is possible. Suppose that 10,000 agents flip coins. Each round, they guess whether a coin that will be flipped is a head or a tail. All agents pay p dollars to make a bet. After the coin is flipped, the amount of money collected from the fees will be distributed among those who correctly predicted the coin flip.

How do we implement this model in NetLogo? Each tick the agents first pay a fee to the pool and determine their guess by flipping a coin. Flipping a coin is modeled by random 2, which provides values 0 and 1 with equal probabilities.

```
ask turtles [
  set earnings earnings - price
  set guess random 2
]
```

The size of the pool is equal to the fee times the number of agents, and the outcome of the coin flip is defined by random 2.

```
let pool nr-agents * price
set coinflip random 2
```

Now for all agents we check whether they guessed correctly the outcome of the coinflip.

```
ask turtles [ifelse guess = coinflip [set correct? 1][set c
```

Those agents who correctly predicted the outcome of the coin flip earn an equal share of the pool. We test whether the number of agents, who made correct predictions, is larger than zero to avoid the unlikely possibility that we divide by 0, which is not allowed.

```
let poolshare 0
if (count turtles with [correct? = 1] > 0) [set poolshare p
ask turtles [
  if correct? = 1
  [set earnings earnings + poolshare]
]
```

So what is the outcome we can expect? On average the earnings will be zero. There is no money coming into the system. As we can see in the simulation of 10,000 agents after 10,000 ticks, the distribution looks like a Gaussian distribution.

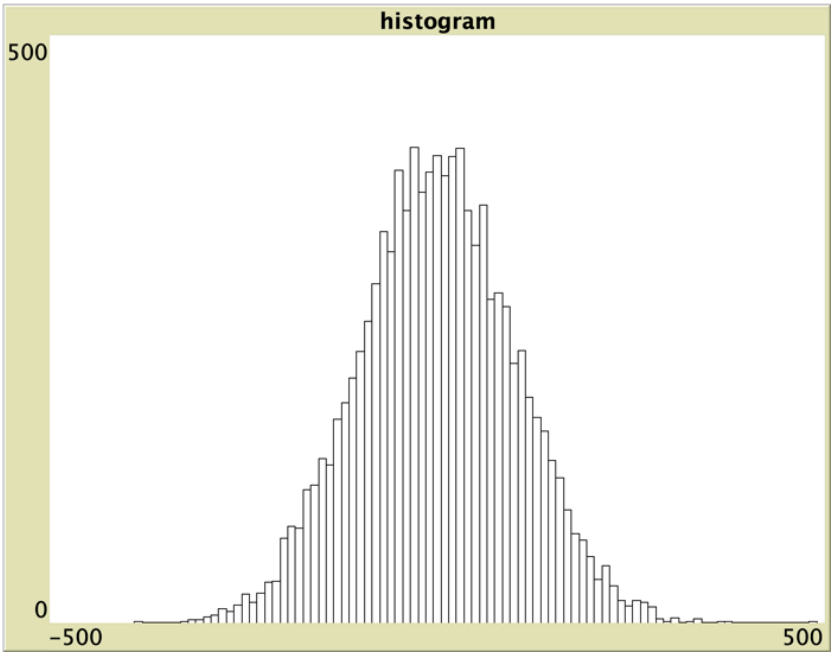

Figure 2. Distribution of earnings after 10,000 ticks.

However, if we look at the maximum earnings of agents in the population, we see that this maximum keeps going up. It does not go up monotonously since lucky agents may experience some rounds of not winning, which reduces the level of the earnings. Nevertheless, Figure 3 shows that at least one of the agents is very lucky and earns a high amount of earnings.

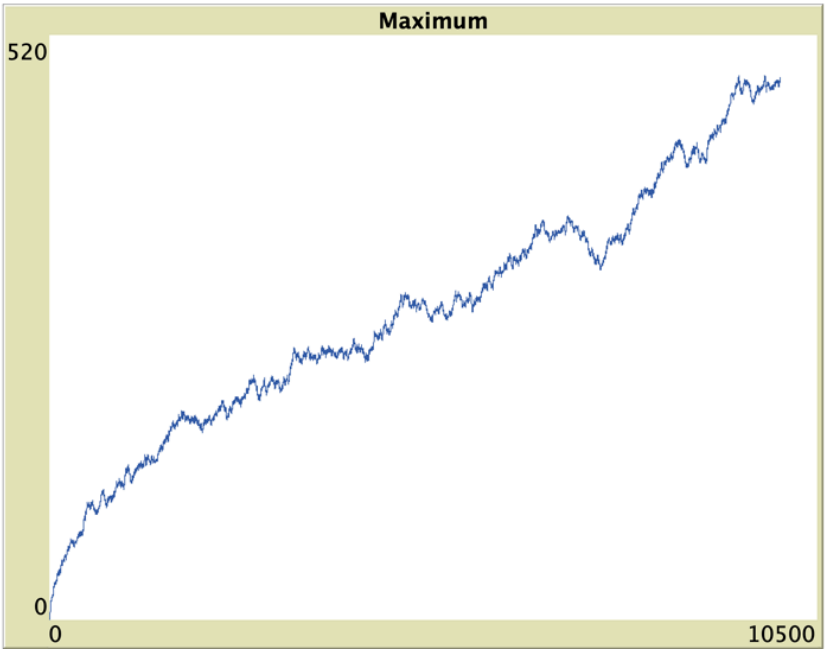

Figure 3. Maximum earnings in the population.

The fact that one can earn a lot by chance is the principle of lotteries and casinos. They charge a small fee to organize those events, say 1% of the contribution to the pool, and if we implement this, the organizers will get 1% of 0.1 dollars of the 10,000 bets per round, which is 10 dollars per round. At the end of 10,000 ticks, the best earner earned something like 300 dollars, but the organizers earned 100,000 dollars. Obviously, the model makes some strong assumptions, such as losers staying in the game. It is not surprising casinos provide free drinks and other amenities for the gamblers to stay and continue to play since the real winner of all these bets is the organizer.

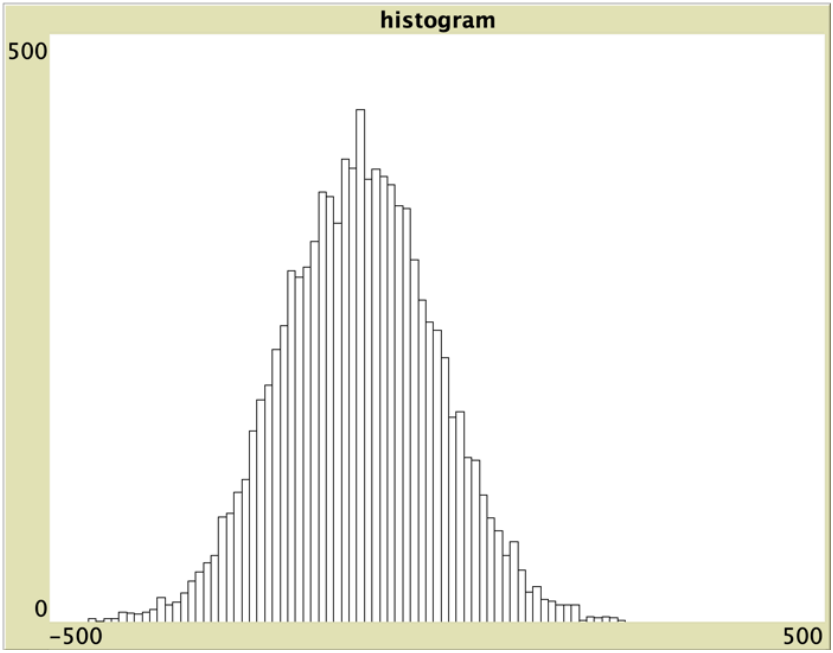

Figure 4. Distribution of earnings after 10,000 ticks when organizers get a 1% fee.

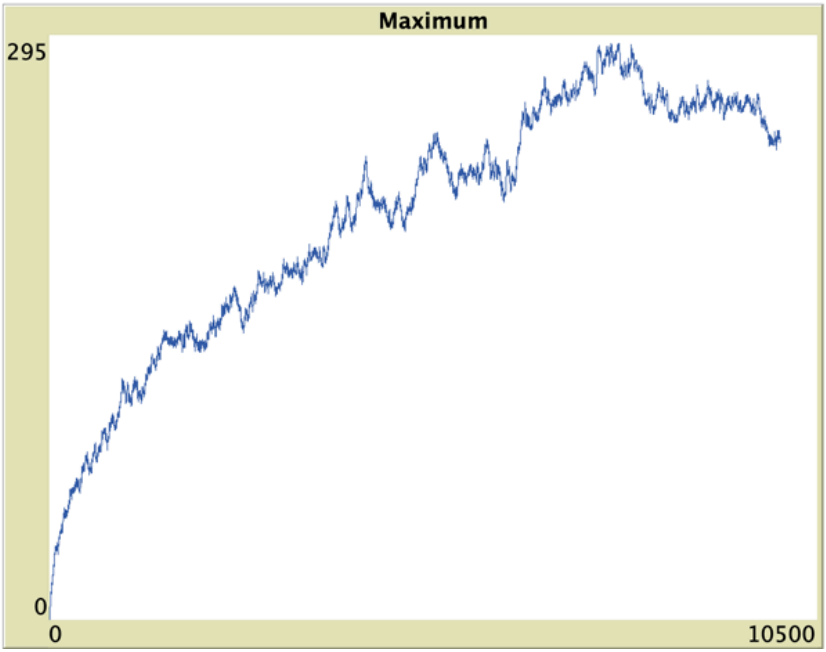

Figure 5. Maximum earnings in the population when organizers get a 1% fee.

Random Walks

There are different ways to model the movement of agents on a landscape. Now we present a simple movement model and show the consequences of different assumptions.

The model is as follows. An agent moves straight with probability probstraight, and if it does not move straight, it will turn right with a random number of degrees, and it will turn left with a random number of degrees. Thus, 2 parameters define the movement: probstraight and degrees.

The basic code in NetLogo looks like.

```
ifelse random-float 1 < probstraight [
  fd 1
][
  rt random degrees
  lt random degrees
```

```
  fd 1
]
```

Different values of probstraight and degrees can lead to very different behavior, as we see below. We keep track of the patches that the agent visits, so that we can visualize the path of say 1000 ticks.

```
ask patches [ set pcolor scale-color green freq 0 maxfreq]
```

In the table below, you see 9 possible patterns. If degrees is 20 the agent is mainly moving straight, making small adjustments. If degrees is 360 and probstraight is less than 0.5, the agent is moving around randomly in a small area.

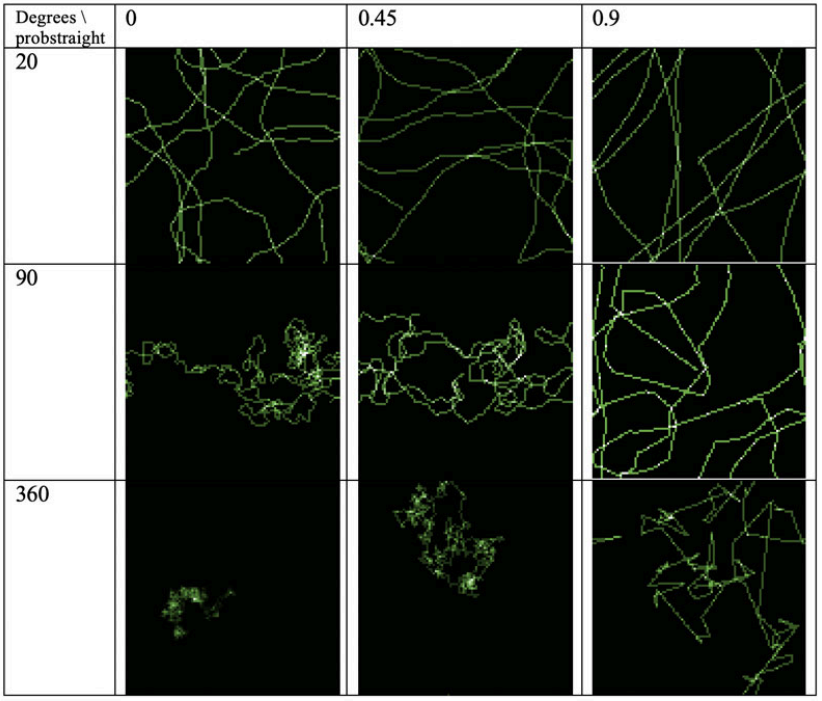

Figure 6. Spatial movement patterns dependent on 2 parameters.

Do different movement rules impact the outcomes of a model? To illustrate this we implement the movement rules described above in the BugHuntCoevolution model which can be found in Curricular Model\BEAGLE Evolution of the NetLogo Model library. In this model, we have bugs and birds. Bugs move around at different speeds, and slow bugs are more likely to be caught by a random bird. As a consequence, we can expect that evolutionary pressures will increase the evolution of faster bugs, which will stimulate the evolution of faster birds.

We replace the code of movement in the procedure wiggle with the random movement rule used above, which depends on the values of degrees and probstraight. We will subsequently test the impact on the evolution of the speed of bugs and birds. We run the model 500 times for 10,000 ticks for several different combinations of parameter values for degrees and probstraight and see how this impacts the evolved speed. Note that we had the same parameter settings for both the birds and the bugs. Figure 7 shows that when bugs move around more locally (high degrees and low value of probstraight), the average speed goes up. If they would not move fast, those parameter settings will keep the bugs at the same location and be a target by the birds. Hence when movement rules lead bug to have more local movements, those who make bigger steps will have an evolutionary advantage. When probstraight is equal to 1, and agents thus keep moving in straight lines, we do not see an evolutionary advantage of going faster. When bugs move locally, it is not an evolutionary advantage to go very fast, and therefore we see the speed of birds dropping a bit when the probstraight is low, and degrees is high.

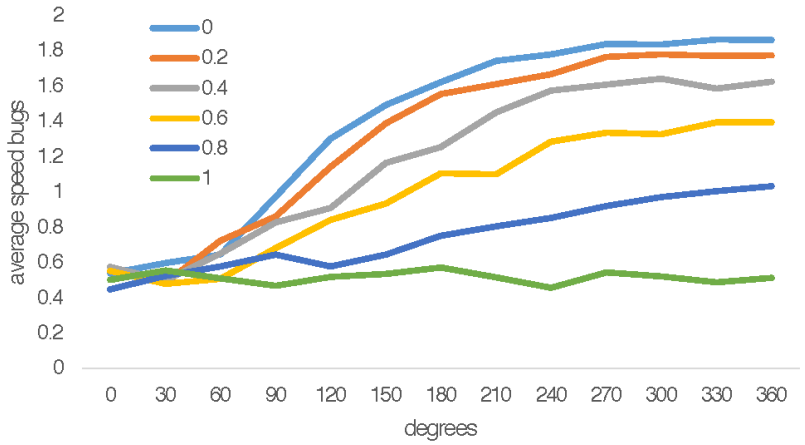

Figure 7. Average speed of the bugs for different values of the movement parameters degrees and probstraight.

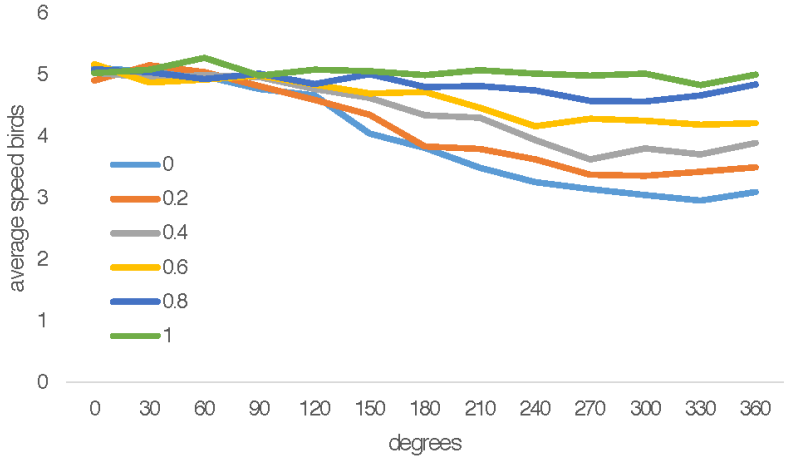

Figure 8. Average speed of the birds for different values of the movement parameters degrees and probstraight.

A next step that one could explore is to allow the movement

parameters degrees and probstraight to evolve over time. What would be the evolved movement patterns? Do the movement parameters always evolve to similar values, or do we see some distinct alternative configurations? Are the evolved movement patterns different for birds and bugs?

Alternative Distributions

So far, we discussed random and random-float, which are uniform distributions. NetLogo support various other distributions.

Normal distribution

The normal distribution is a widely applied distribution and is defined by the mean and standard deviation.

```
random-normal average stdev
```

The distribution produces values from minus infinity to plus infinity and is symmetric.

Poisson distribution

The Poisson distribution is defined by the mean and produces positive integers. It is typically used to express the probability of a number of events in a certain interval, such as the number of emails received during the last hour.

```
random-poisson mean
```

Gamma distribution

The gamma distribution is a very general distribution that can produce a quite diverse range of shapes of the distribution dependent on the values of alpha and lambda. It produces positive real numbers.

```
random-gamma alpha lambda
```

Exponential distribution

The Exponential distribution is defined by the mean and is a probability distribution that describes the process in which events occur continuously and independently at a constant average rate.

It is a special case of the gamma distribution and produces positive real numbers.

```
random-exponential mean
```

Summary

Most agent-based models are stochastic simulation models. Therefore, we provide a number of examples in this chapter to demonstrate the different ways to implement stochastic processes in NetLogo to generate stochastic events in time and space.

CHAPTER 12

Sensitivity Analysis

Key Concepts

In this chapter, we will:

- get familiar using BehaviorSpace to run a NetLogo model,
- experience how to use BehaviorSpace to do a sensitivity analysis of a model,
- learn how to evaluate how many runs are sufficient to get reliable statistics for model analysis.

With sensitivity analysis, we refer to the analysis of the outcome of the agent-based model for different assumptions. This could be different parameter values for different functional forms defined for specific mechanisms in the model. The reason for doing this analysis is to derive a better understanding of the model and how robust the model results are to alternatives within the range of uncertainty, to obtain a better understanding on the relationship between different assumptions, to identify areas of the sensitivity of the model, and identify components of the model that could be simplified.

To do a proper sensitivity analysis, we first demonstrate the

NetLogo tool BehaviorSpace, which will automate the process of running many simulations and store data in a csv file. We discuss how to identify the minimum number of repetitive model runs sufficiently to get reliable statistics of the variability of the outcomes of an agent-based model.

How to Use BehaviorSpace

BehaviorSpace is a tool of NetLogo to automatically run a large number of simulations and collect the data that you are interested in in a text file and/or Excel file. BehaviorSpace can be found in NetLogo under Tools.

Before we go further, it would be useful to get a model from the library to run some experiments for an actual model. We will select the basic traffic model from the NetLogo library (Social Science -> Traffic Basic):

This model simulates a number of cars that have each a simple set of rules. The car moves from left to right with a certain speed and slows down if it sees a car close ahead, and speeds up again if it doesn't see a car ahead. The model has two parameters, deceleration, and acceleration, which defines how fast a car reduces speed and speeds up. The car also has a minimum speed (=0) and a maximum speed (=1). When we run the model, we see that traffic jams emerge. Figure 1 shows the simulated environment where the cars that exit the road at the right will enter at the left again. As such, a group of cars is driving in circles, and traffic jams emerge.

Figure 1: Screenshot of the Traffic model. The red car is the focal car in the visualization of the results.

When do we get traffic jams without accidents or other obstructions? Since cars react to each other and have different

speeds, a slower car than the one behind makes the car behind decelerate. Since deceleration and acceleration are not instantaneous, the interactions between cars lead to the emergence of groups of cars with slow speed. Figure 2 shows that at the beginning of the simulation, there is some irregularity, and after about 200 ticks, the car starts accelerating and decelerating predictably. The first part of the simulation is what we call a warm-up period. The initial conditions (initial location and speed of cars) define the results until we get a pattern defined by the parameters of the simulation. Figure 3 shows the average speed of cars and has a similar pattern.

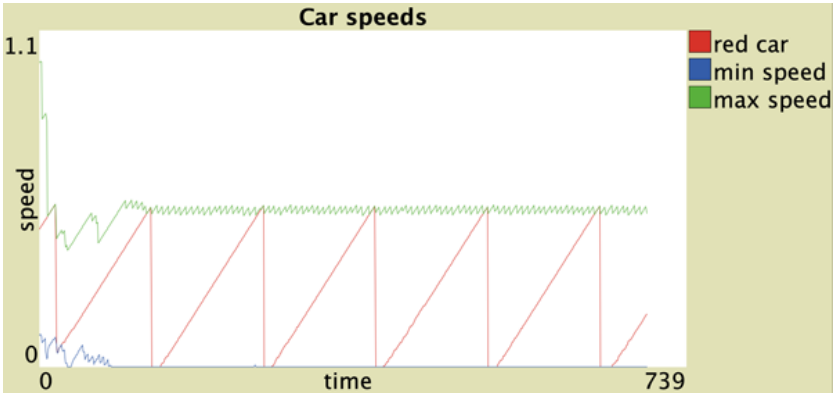

Figure 2: The speed of the red car goes up and down in the simulation.

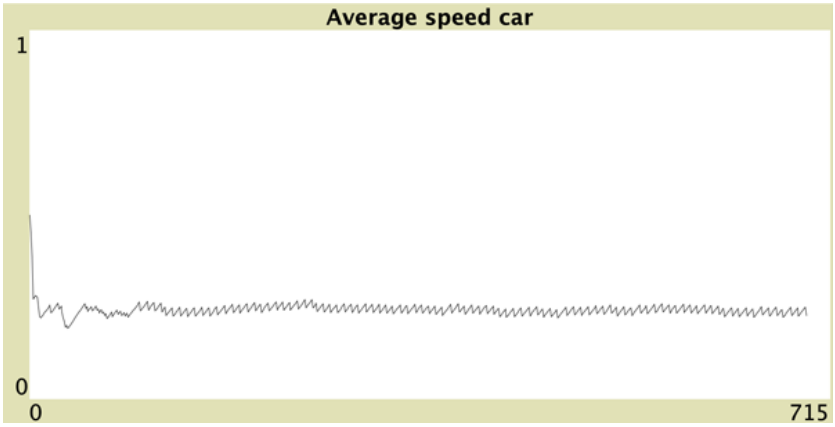

Figure 3. The average speed of 20 cars during the simulation.

Suppose we want to explore which values of deceleration and acceleration the speed of the cars are high we have to run the model for different parameter settings and measure the average speed of the cars after say 1000 ticks. Since there is stochasticity in the model, we will run the model multiple times, say 100 times for each parameter setting.

Instead of doing this manually, we will use BehaviorSpace. Go again to BehaviorSpace in "tools." When the following screen pops up:

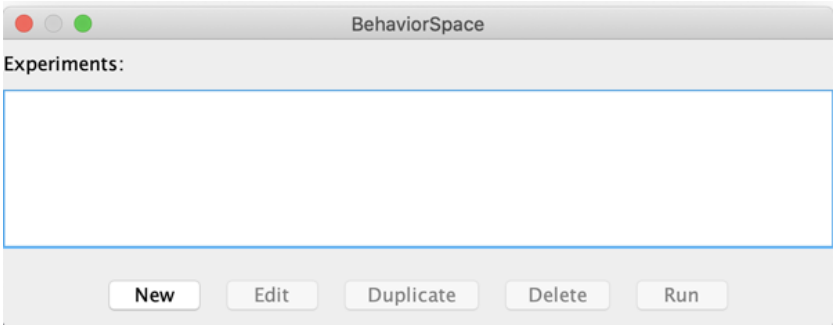

Figure 4. Screenshot of the BehaviorSpace pop-up window.

Click on “new.” We will now generate the settings for running an experiment. In Figure 5 we see that the global variables (including the sliders on the screen) are listed in the list of variables you can vary. The values given are the default values. We also see the number of repetitions we want to run the model for each parameter setting.

Then we see the information we will save in a text file or Excel file. The default is “count turtles.”

We can store information for each time step or only the results at the end of the simulation (Measure runs at every time step).

The setup commands define that the command setup will setup the model.

The go commands define that the command go will let the model run forever.

The time limit is the maximum number of ticks that we will allow the model to run.

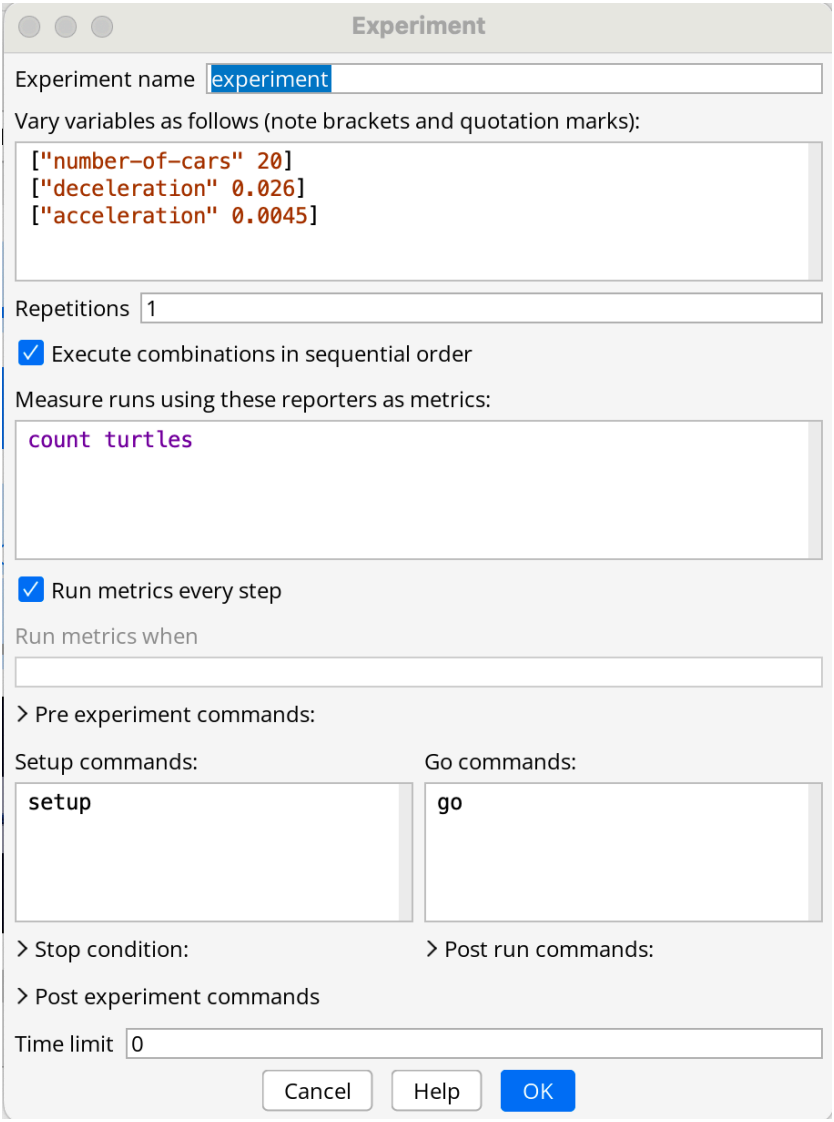

Figure 5. Screenshot of the settings of an experiment.

Now we want to run a series of experiments by varying the level of

deceleration and acceleration used by all agents. We will vary the variables in the following ways:

```
["acceleration" [0.0025 0.0005 0.0075]]

["deceleration" [0.014 0.003 0.044]]
```

This means that we explore the model for values of acceleration from 0.0025 to 0.0075 with steps of 0.0005. Hence there will be 11 different values. Similarly, we use 11 different values for deceleration. This leads to 11 x 11 = 121 different parameter combinations. Each parameter combination will be run 100 times (100 repetitions). This means we will run a total of 12,100 simulations of the model.

Now we have to define what information to save. The default case is the number of turtles, which is not very useful in our analysis since it remains constant. More interesting will be the average speed of all cars. Therefore we need to save mean [speed] of turtles

We don't want to save the information for every time step since we are interested in the average speed at the end of the simulation. Therefore, we should uncheck Measure runs at every time step. This will also speed up the simulations since data will not be saved every tick.

We want the model to run 1,000 ticks, so we select a time limit of 1000.

Now we get something like

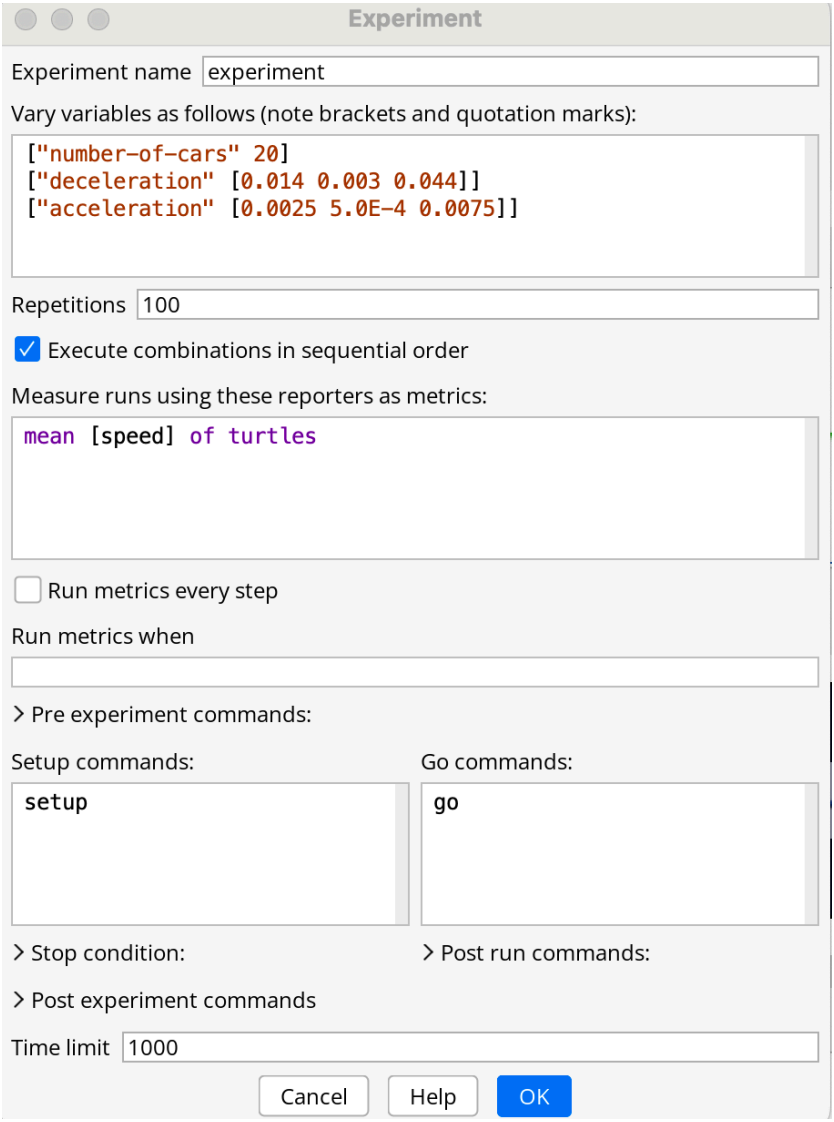

Figure 6. Screenshot of the parameter settings of BehaviorSpace.

Now we click OK and get

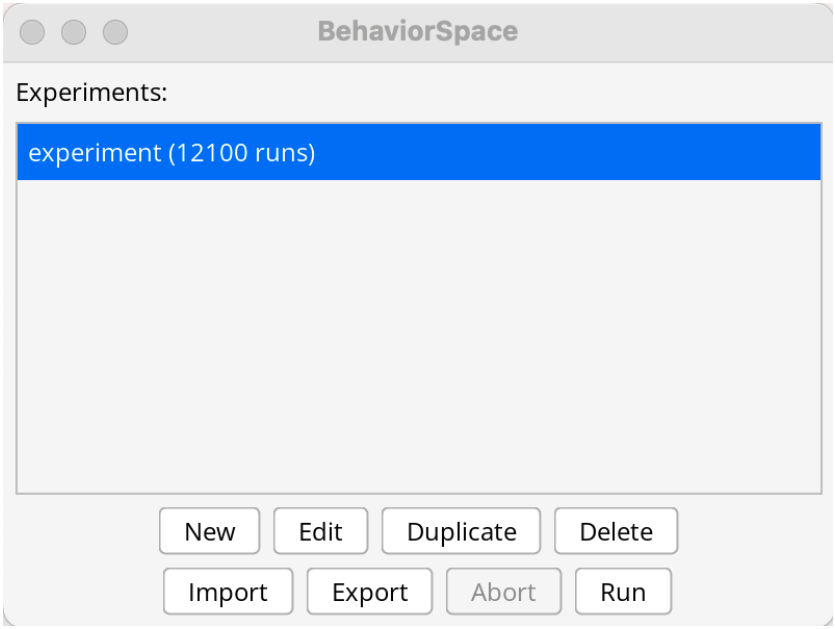

Figure 7: Screenshot popup experiments after experiments have been defined.

The value 12100 means that we have defined 12,100 runs. Note that we run each parameter settings 100 times, and therefore we have 121 different parameter settings. For this exercise, we want to keep the runtime short.

Now we have to click on “Run” and get the following:

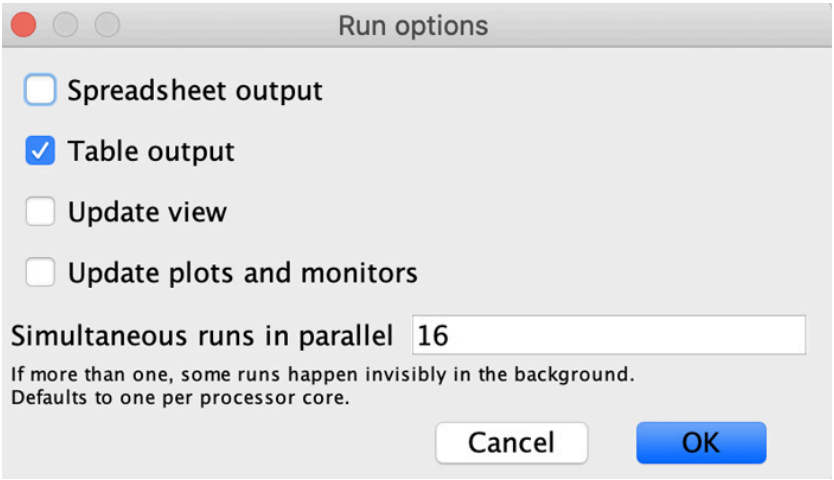

Figure 8. Screenshot of run options.

We have to decide to save it as a Spreadsheet output (Excel), a Table output(ascii text file), or both. We recommend you to use the Table output option but urge you to check out both ways of saving the data. When we click on one of the options, we are asked where to save the data on the computer. When we have done this, the computer runs the simulations and stores the data. We will then get the following window:

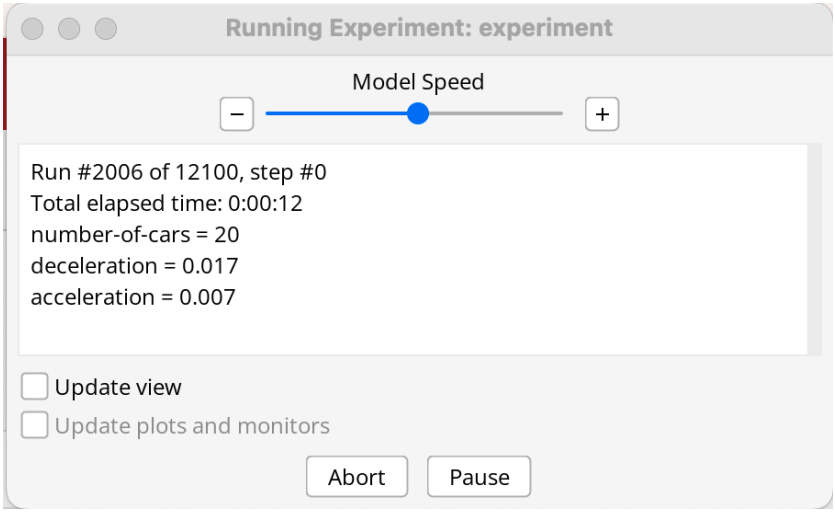

Figure 9. Screenshot of the experiment in action.

We see here how many runs it has done so far. We speed up the runs by unchecking the "Update view" and "Update plots and monitors" so that the computer does not spend time to update the screen for each calculation. And we can put the slider to high speed instead of normal speed. For this exercise, it is not very important, but for more complex models, we may run BehaviorSpace for many hours, and then it is useful to reduce the time for the calculations by 90%.

Now we can look at the results. By importing the data from the table file into Excel, we get the following picture. We may check the differences in how the information is stored between the spreadsheet and the table. I prefer to use the table for my analysis.

	A	B	C	D	E	F	G
1	BehaviorSpa	Table version 2.0					
2	Traffic Basic.nlogox						
3	experiment						
4	06/26/2026 10:34:07:368 -0700						
5	min-pxcor	max-pxcor	min-pycor	max-pycor			
6	-25	25	-4	4			
7	[run number]	number-of-c	deceleration	acceleration	[step]	mean [speed]	of turtles
8	9	20	0.014	0.0025	1000	0.68578483	
9	2	20	0.014	0.0025	1000	0.96561478	
10	5	20	0.014	0.0025	1000	0.69697503	
11	6	20	0.014	0.0025	1000	0.77162535	
12	12	20	0.014	0.0025	1000	0.77346211	
13	11	20	0.014	0.0025	1000	0.93794803	
14	7	20	0.014	0.0025	1000	0.81411655	
15	8	20	0.014	0.0025	1000	0.71823379	
16	10	20	0.014	0.0025	1000	0.88627813	
17	4	20	0.014	0.0025	1000	0.76803964	
18	1	20	0.014	0.0025	1000	0.77917128	
19	3	20	0.014	0.0025	1000	0.73947791	
20	14	20	0.014	0.0025	1000	0.73689333	
21	13	20	0.014	0.0025	1000	0.75607498	
22	15	20	0.014	0.0025	1000	0.90556372	

Figure 10. Screenshot how the data is stored in the Table format. For each run, the parameter values and output is listed.

We can now use Excel to create a Figure. A useful tool to process the data is the PivotTable in the Insert tab. When we create a figure of the average speed of the 100 runs for each parameter combination, we get a graph like Figure 11. We see that we can increase the average speed of the cars to the maximum level if the deceleration rate is low and the acceleration rate is fast. After some initial hick-ups, the cars end up being within sufficient distance from each other, not to reduce speed. If cars reduce speed quickly when they see a car in front of them, this leads to traffic jams. Notice a critical assumption is that all cars have the same acceleration and deceleration rates. Additional analysis could be done to explore the consequence if there is some variation of those rates between the cars.

We used Excel to make the figures and do the analysis, but one can use other data analytics tools, such as R scripts, to analyze the csv files that BehaviorSpace produces. If you are familiar with R, try out to replicate this analysis and use R for creating the figure.

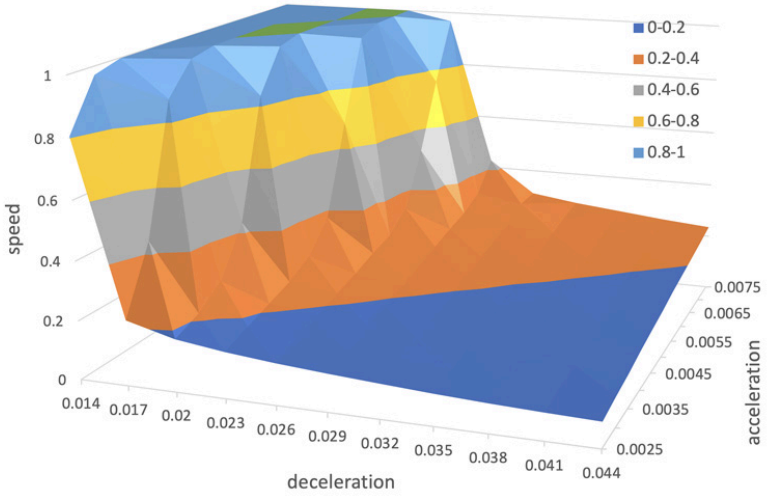

Figure 11. Relation between the average speed of cars for different rates of deceleration and acceleration.

More information on BehaviorSpace can be found at http://ccl.northwestern.edu/netlogo/docs/behaviorspace.html.

How Many Runs?

Agent-based models do often include stochastic processes, and as such, lead to different outcomes each time you run a model. If we want to derive insights from the model, we need to do sufficient runs of a certain parameter setting to get an idea of the variation of the outcomes of the model. If we want to compare whether a different variation of the model leads to significantly different outcomes of the model, we need to know whether we have run the model sufficient times to make such a comparison.

Since an agent-based model can be formulated in many different ways, there is no specific approach to determine the sample size,

the number of runs, in order to have outcomes with a certain precision. This is in contrast to other stochastic methods, like Markov processes, which are more constraint in their formulations.

Although there is subjectivity involved in determining a sufficient number of runs to get simulated data that you can use to compare reliably different outcome distributions, we can apply some formal approaches. Based on Lee et al. (2015), we present a method that does not assume normal distributions underlying the agent-based model.

The first method is based on the variance of the simulated outcomes and aims to have sufficient runs to derive a stable variability.

We first calculate the coefficient of variation as the ratio of the standard deviation of a sample (σ) to its mean (μ):

$c_v = \sigma / \mu$

The coefficient of variation can vary quite a bit for small sample sizes. When we run a model many runs, we will evaluate the variability of c_v. We define a criterion, an error margin, E. If the difference between consecutive values of c_v for increasing values of the sample size stays below E, we have found our minimum number of simulations. A typical value for E could be 0.01.

Hence the minimum number of runs, n_{min} is defined as

$n_{min} = \text{argmax}_n \mid c_v^{x,n} - c_v^{x,m} \mid < E, \forall x \; \& \; \forall m > n$

where n_{min} is the estimated minimum sample size; x is a distinct outcome of interest, and m is some sample size >n for which the c_v (of each outcome) is measured.

We illustrate the calculation for the minimum number of simulations with the model Traffic Grid in the Social Science folder of the NetLogo library. The model is an extension of the Basic Traffic model we discussed earlier, but now we put cars in a grid of roads, where cars experience stopping lights. Cars will slow down if there is a car in from of them or a red stop sign and speed up to maximum speed if there are no cars in from of them. Figure 12 shows that traffic jams emerging in this city grid. We will explore

with a sensitivity analysis whether there are ways to improve the cars' average speed.

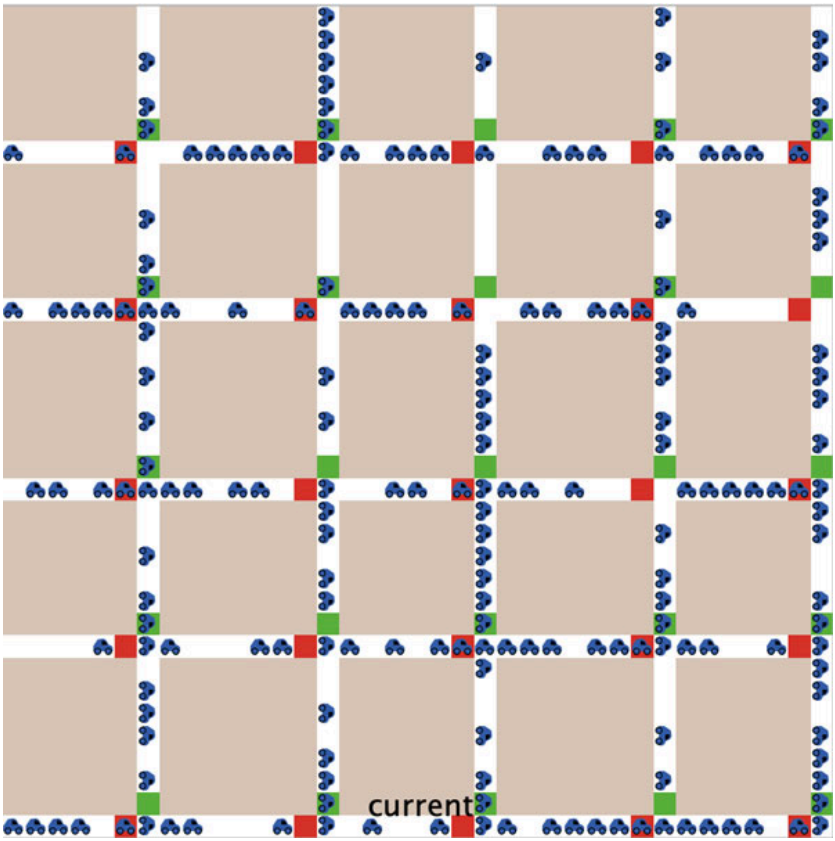

Figure 12: Screenshot of Traffic Grid.

First, we need to determine how to evaluate the performance of the system. Figure 13 shows the average speed of cars during a simulation of more than 5000 ticks. It shows that there is quite some variability. We propose to calculate the mean speed between 1000 and 1100 ticks so that there are no start-up effects, and a mean of 100 ticks allow us to have a better estimate of the

system, compared to just one tick, given the variation of speed in the system.

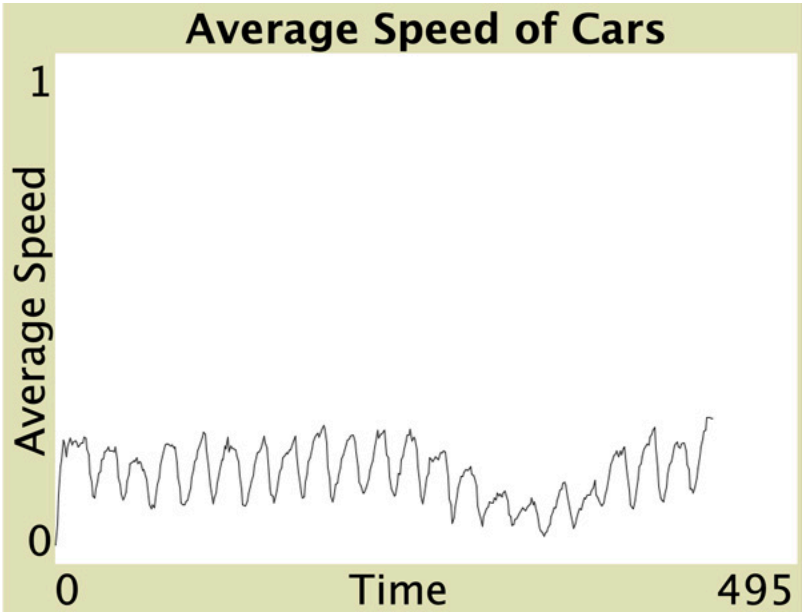

Figure 13: The average speed of the 200 cars over time.

We run the model 400 times and calculate the standard deviation and mean for an increasing number of runs. This allows up to calculate the coefficient of variation (Figure 14). We find that after about 50 ticks, the change is dropped for the first time below 0.001. As such, we decided that the minimum number of runs needs to be 50 runs. In sum, the analysis shows that about 50 runs are needed to have the desired accuracy of the output metrics. It is good to realize that this number depends on the specific parameter settings used to calculate the minimum number of runs.

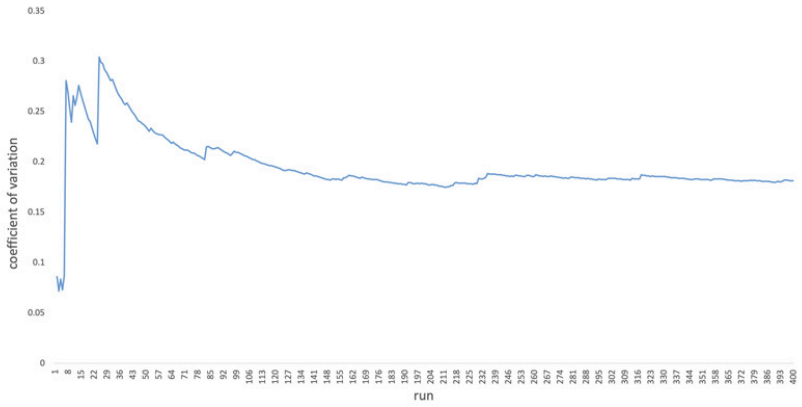

Figure 14: Coefficient of variation (cv) for an increasing number of runs.

As a sensitivity analysis, we calculated the average speed for different numbers of cars in the system. For each speed, we run the model 50 times. Figure 15 plots the average speed for the cars and adds an error bar with the size of the standard deviation for the 50 runs. We see that for a smaller number of cars, the average speed of the cars goes up.

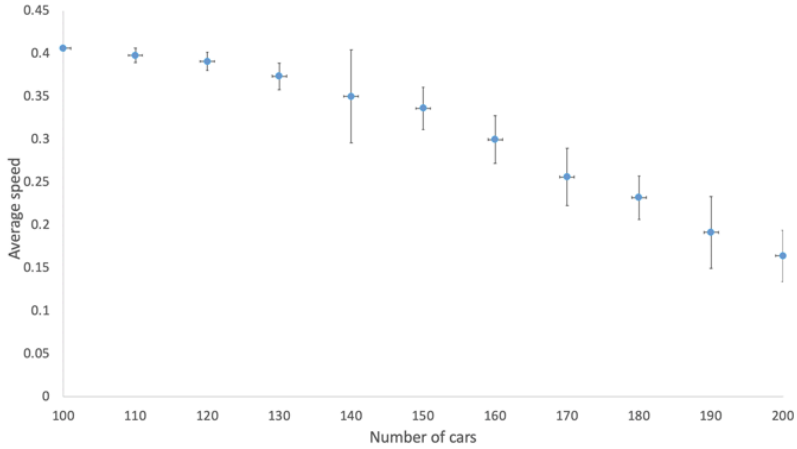

Figure 15: The average speed of the cars changes when the number of cars in the system changes.

The sensitivity approach we discussed above is called One-at-the-time, where we vary one parameter systematically and evaluate the outcomes. There are many more advanced methods possible. One simple extension is to vary 2 parameters systematically, as we did for Figure 11. We could also sample the parameter space and estimate a statistical model of the parameters you varied and the outcome. In this way, you use the simulation model to generate data to estimate a statistical model. Finally, one can use something like variance-based sensitivity analysis, which allows us to estimate the relative contributions of the different parameters to explain the variance in the outcomes.

A good reference to learn more about sensitivity analysis for agent-based modeling is Lee et al. (2015).

Summary

The chapter provided a tutorial on how to use BehaviorSpace, which could be used for sensitivity analysis. Furthermore, we provided guidance on the number of simulation runs to use in a sensitivity analysis.

References

Lee, Ju-Sung, Tatiana Filatova, Arika Ligmann-Zielinska, Behrooz Hassini-Mahmooei, Forrest Stonedahl, Iris Lorscheid, Alexey Voinov, Gary Polhill, Zhanli Sun and Dawn C. Parker (2015), The Complexities of Agent-Based Modeling Output Analysis, *Journal of Artificial Societies and Social Simulation* 18 (4) 4 <http://jasss.soc.surrey.ac.uk/18/4/4.html>

Lorscheid, Iris, Bernd-Oliver Heine, and Matthias Meyer (2012). Opening the 'black box of simulations: Increased transparency and effective communication through the systematic design of experiments. *Computational and Mathematical Organization Theory* 18, 22–62

If you want to see examples of using R to analyze outputs of NetLogo models, you may check out this GitHub repository with examples for the WolfSheep model.

CHAPTER 13

Comparing simulated data with empirical data

Key Concepts

In this chapter, we will:

- discuss how to evaluate a simulation model with empirical data,
- introduce BehaviorSearch for optimization tasks,
- demonstrate how to use BehaviorSearch to find the best parameter values for the calibration of an agent-based model,
- use Approximate Bayesian Computation to find distributions of the parameter values that explain the empirical observations best.

How do we know whether our agent-based model is a good representation of reality? There are many types of data, and many ways to evaluate model outcomes. In this chapter, we will demonstrate a few conventional ways to compare simulated results with data and adjust model parameter values to get a better fit.

But first, we need to discuss some of the underlying challenges. When we develop a model, we made choices on what variables and processes to formalize. Perhaps we select those variables because there is quantitative information available. We have to realize that by making a formal model, we simplify reality, and often make choices to favor quantification. Hence when we evaluate whether the model generates data similar to empirical data, we have to realize that we have made choices on what the model represents and what the important variables are.

In this chapter, we will first discuss 3 common ways in agent-based modeling to evaluate model outcomes with empirical data. Then we demonstrate BehaviorSearch, a computational tool that you can use to explore the parameter space of a NetLogo model, for example, by using optimization tools to find the best fit between data and the simulations. Finally, we apply the three approaches for calibration to the Artificial Anasazi model.

Three ways to evaluate a model fit with data

A common way to evaluate the performance of a model is to compare the simulated (s_i) and empirical data (e_i), for example, by the sum of $\sum_{i=0}^{n} (s_i - e_i)^2$. A lower value of this sum is a better fit between simulated and empirical data. Having a good fit does not mean that a model is correct. It is possible that different mechanisms lead to similar outcomes (equifinality). For complex systems, we may also look at a number of different variables, but how do we weigh the different variables? Are they all equally important?

A standard approach for evaluating a model is to use optimization to find the parameter values that minimize the difference between the simulated and the actual data. Since agent-based models are often stochastic models, finding a global optimum might be a challenge. We could use computational optimization tools like genetic algorithms. Still, we have to realize the derived "optimal" parameter values cannot be guaranteed to

be the best fit with the restricted time we have to explore the parameter space.

Grimm et al. (2005) propose Pattern-Oriented modeling in which you define a number of patterns in the data and an uncertainty area around the pattern. Model versions that generate simulated data within the uncertainty boundaries of all patterns are considered to be equally valid. Hence you are not looking for the best model version, but the model versions approximate all patterns. The more diverse patterns you consider, the more likely model versions that produce results within the uncertainty ranges are structurally sound. One of the challenges is to define patterns. The patterns could be spatial and temporal patterns, distributions at the population level, metrics of movement, and change. This will be a somewhat subjective matter. The expectation is that more patterns are better since it provides more constraints for potential model configurations. But note that if no model configuration can meet the patterns, the model is not considered to explain the data. What is the tolerance of error? If you have multiple observations, such as experimental data, you can define a statistically based interval. However, for many applications, you have only one case, and as such, you may have to define expert-based tolerance intervals.

A third approach we look into is Approximate Bayesian Computation (ABC). There is a long history of Approximate Bayesian Computation (Beaumont, 2010), and based on Van der Vaart et al. (2016), we discuss the application for doing parameter estimation with rejection-ABC. The following is the basic procedure (Van der Vaart et al., 2016):

– Define prior distributions for all of the model's parameters you want to estimate.

– Run the agent-based model e.g., 100,000 times, using random samples from its prior distributions.

– Accept the e.g., 100 runs that provide model outputs that best fit the empirical data.

– Analyze the accepted parameters to obtain approximate posterior distributions.

Before we discuss an example, we will introduce BehaviorSearch.

BehaviorSearch

BehaviorSearch is a software tool bundled with NetLogo to explore agent-based models in an automated way to find parameter settings of the model that lead to certain desired outcomes. BehaviorSearch uses computational algorithms such as genetic algorithms, random search, simulated annealing, and hill-climbing, to search the parameter space.

When you open BehaviorSearch, you get the screen shown below. You will find a good tutorial of the model if you click on "Help." You can use "Browse for model" to identify the NetLogo model you want to explore. The Parameter Specification includes those parameters and values you have defined on the Interface of NetLogo. "Setup" and "Step" define the procedures the model uses to initialize the model and simulates each tick. "Measure" is the metric one wants to evaluate. This could be the distance between simulated and empirical data that one wants to minimize. It is a global variable that captures outcomes of interest from the model. With "Measure if" you can control for which ticks measure is calculated. For example, we can include "ticks >= 100" to only calculate measure if ticks are 100 or larger.

"Stop if" is an optional condition to stop the model. "Step Limit" is the maximum number of times the procedure defined by "Step" is called.

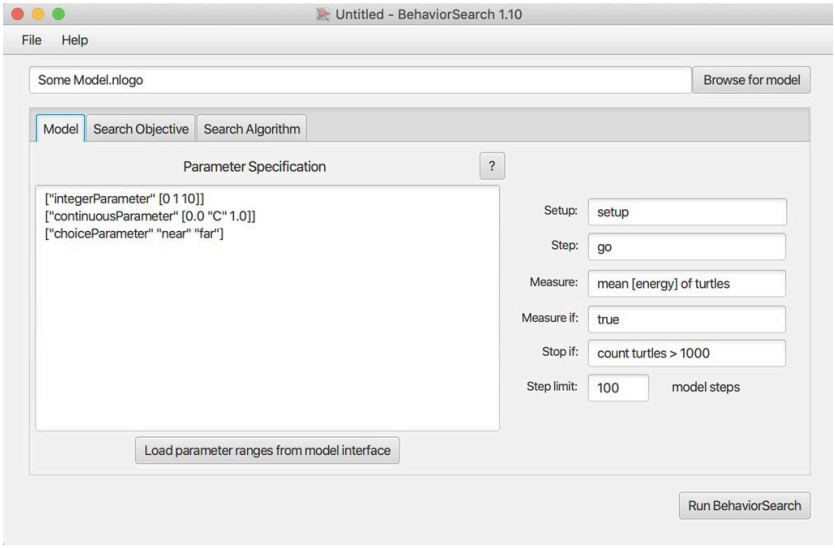

Figure 1. Screenshot BehaviorSearch.

We will now load an example. Use “File” at the left top of the screen and go to the subdirectory app/behaviorsearch/examples of your NetLogo folder and select the Flocking example. We now import the information for this example and see familiar parameters (Figure 2). Note that you need to click on “Load parameter ranges from model interface” to get the right set of parameters. The measure is the standard deviation of the change of position since the model looks for parameter settings that lead flocking agents to converge quickly to all move in the same direction.

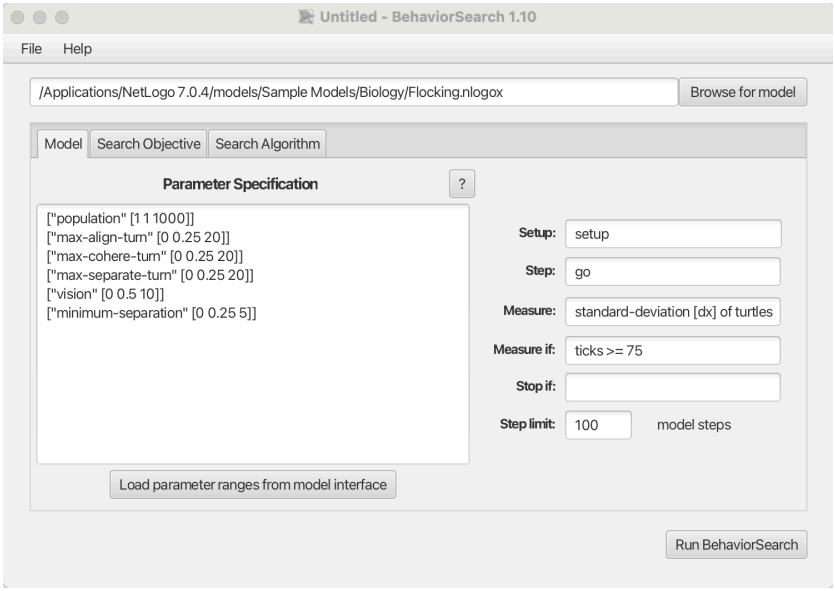

Figure 2. Screenshot imported NetLogo model in BehaviorSearch.

If we click on the tab "Search Objective," we can define what we want to optimize. In this example, we like to minimize the value of the measure, which we can see from the selection of "Minimize Fitness." You could also have selected "Maximize Fitness." There are different options to define fitness. Will this be the value of measure at the last step of the simulation, the mean across all steps of the simulation, the minimum or maximum values during the simulation, the median of all steps, or even the variance. Hence you have many options to define how you want to evaluate the outcomes of the model.

To define the fitness value, do we need to run a model multiple times since an agent-based model may include stochastic elements? And if we run a model many times, how do we aggregate the information into one fitness value? We can use the mean, maximum, minimum, median, variance, and standard deviation of the simulations.

You can select “Take derivative?” if you want to estimate whether you are located in a local optimum, given a certain measure of error. However, this will be an unusual choice for an agent-based model since stochastic models will be difficult to evaluate on their derivatives.

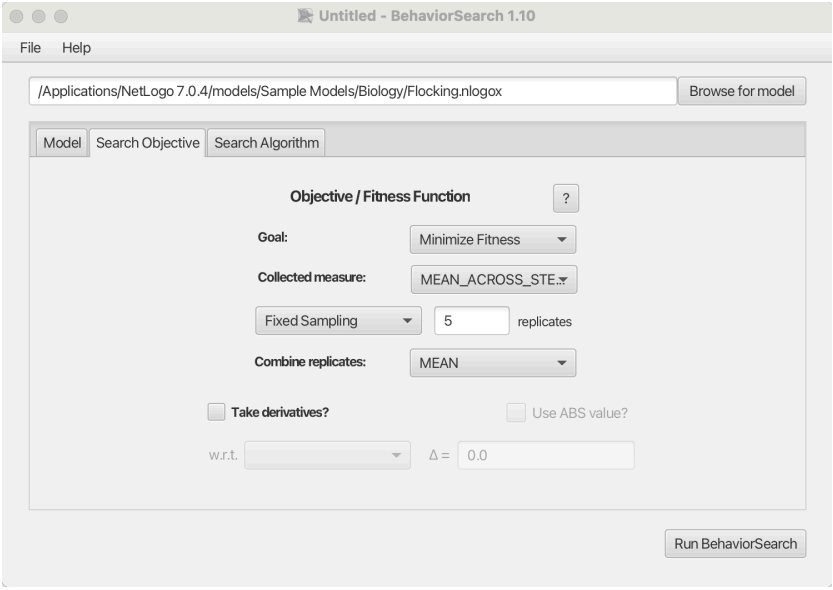

Figure 3. Screenshot Search Objective tab.

If we now click on “Search Algorithm,” you find four different ways to explore the parameter space: Random Search, Genetic Algorithm, Simulated Annealing, and Hill Climbing. Random Search generates random combinations of parameter values and keeps track of the best results. Genetic Algorithm has a population of solutions that evolve to better outcomes, Simulated Annealing and Hill Climbing are both numerical approaches to find local optima.

None of those approaches guarantee the finding of the best fit if we have stochastic non-linear models. We generally use Genetic Algorithm because it is a useful tool to explore non-linear stochastic systems and because we have good experience with it.

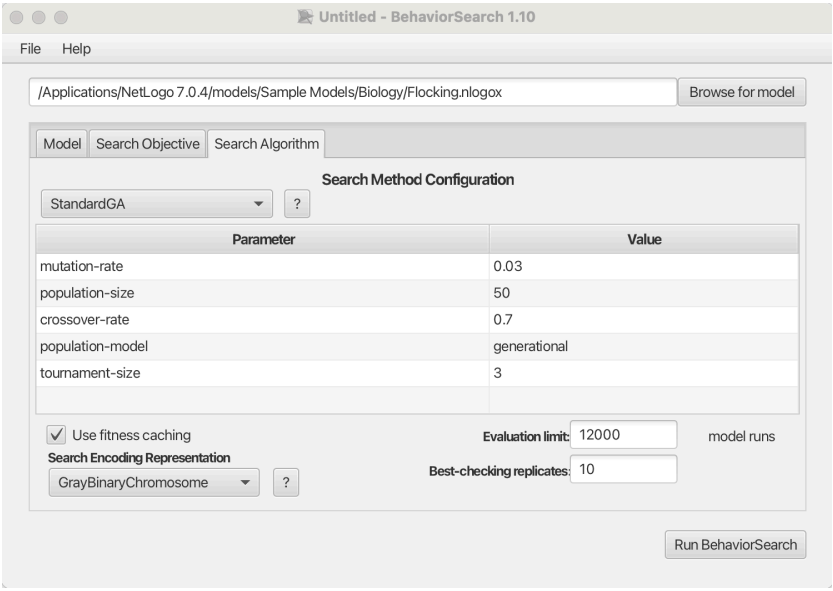

Figure 4. Screenshot Search Algorithm tab.

In Figure 5, you see the results if we click "Run BehaviorSearch." You see the Fitness value over time using the Genetic Algorithm. It shows at the right the parameter settings leading to the best outcome and indicates the expected duration of the search. In this example, we see the fitness value going up, which can happen in a Genetic Algorithm. The new generation of a set of solutions may not contain the best performing solution of the previous generation again. In Hill Climbing type of approaches, you will continue to improve the values of the fitness score. Try out BehaviorSearch and test out what happens with the different search algorithms.

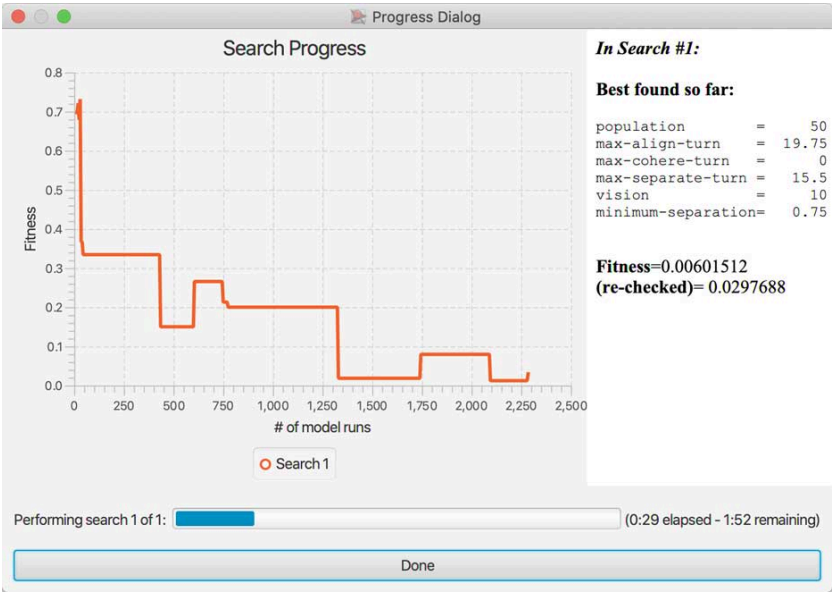

Figure 5. Screenshot Search Progress of a genetic algorithm search.

Artificial Anasazi Example

We will demonstrate three approaches, traditional calibrations, Pattern-Oriented Modeling, and Approximate Bayesian Computation, with the Artificial Anasazi model, which can be found in the NetLogo library (Social science -> unverified models -> Artificial Anasazi). Axtell et al. (2002) developed this model to understand whether the depopulation of Long House Valley, in Arizona, USA, around 1300 can be explained by persistent drought or whether other factors might be in play. The model runs from 800 to 1350 in annual time steps. This is the period in which the population in the Valley increased to about 250 households, after which it declined. After 1300 there is no evidence of new constructions of houses in the valley. Although we use the data as precise estimates of population numbers, the reality is that they are based on guestimates of excavated sites. Although we do not know how the excavated buildings were used, estimates of occupation

are made using room counts. For each of the sites, there are estimates of occupation duration and a possible number of occupants. Hence there is a lot of uncertainty within the empirical data, but in our analysis, we will use the data provided as the true empirical observations we want to explain. The reason is that this data available is the best we have available, but the uncertainty in the data urges us to interpret the results with care.

Each agent represents one household consisting of 5 persons. The composition of households is not changing over time. Each household generates a demand for a certain level of food. To supply this demand, the agent performs agriculture. The cell on which an agent farms is different compared to the cell an agent occupies for its house. Each agent has its unique cell for farming, but more than one agent can have a house on a cell. There is no sharing of food between households, and the storage of food is at the household level. Each household makes annual decisions on where to farm and where to locate the house. A household has an age and a stock of food surplus from previous years.

A typical simulation of the default settings shows the rise and fall of the population, but not the complete abandoning of the site since the simulated households are still positive after 1300 (Figure 6). In fact, there is still productivity of the land that could feed households.

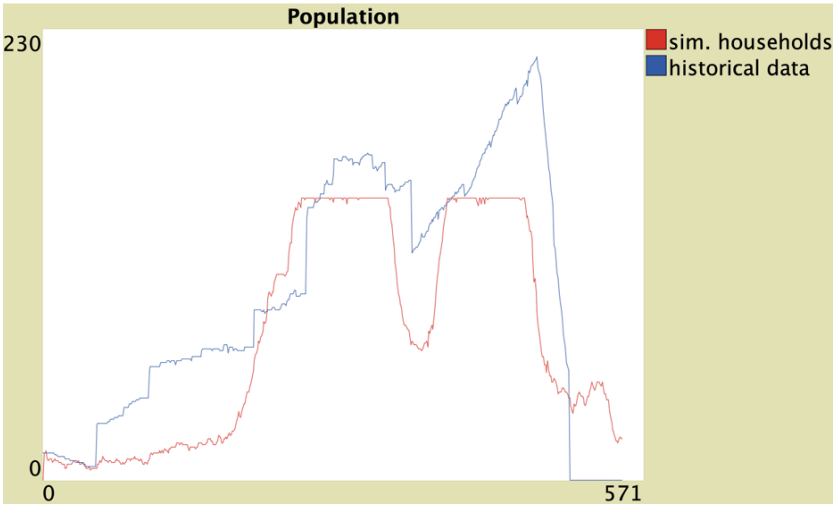

Figure 6. A typical run of the Artificial Anasazi projecting simulated and observed household levels.

We will also show that the outcomes could vary quite a bit for the parameter settings of the model used to determine the minimum number of runs. Axtell et al. (2002) used 15 simulations, but Figure 7 indicates that this leads to a frequent change of more than 0.01 in the coefficient of variation. We decided to use 100 runs for each parameter setting. In order to evaluate the outcomes, we look at the median of the squared difference, not the mean, due to a small probability of a very bad fit. This is demonstrated in Figure 8, where we find a number of outcomes from a particular parameter setting leading to a very large value of L2, the value of the difference between the data and simulation. This is caused by not generating the rise of the population levels as observed. Since those bad runs heavily weight on the mean of L2, it is better to use the median of L2 to be comparable among parameter settings.

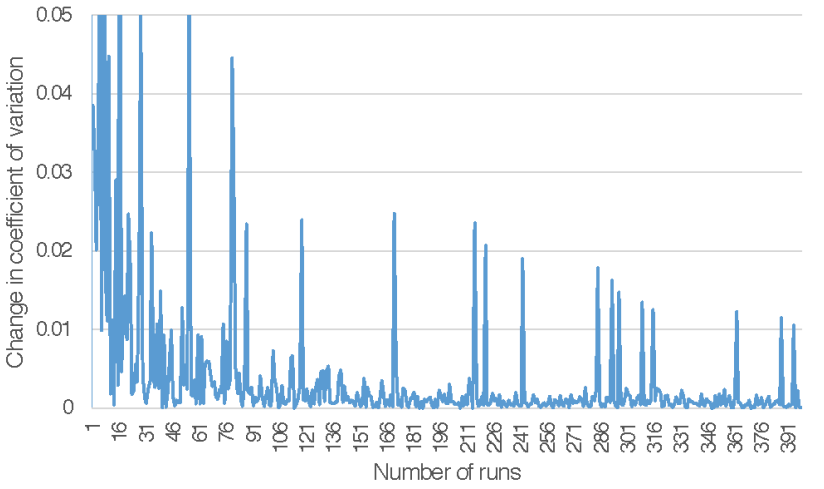

Figure 7. Change in the coefficient of variation for 400 simulations of the default Artificial Anasazi model.

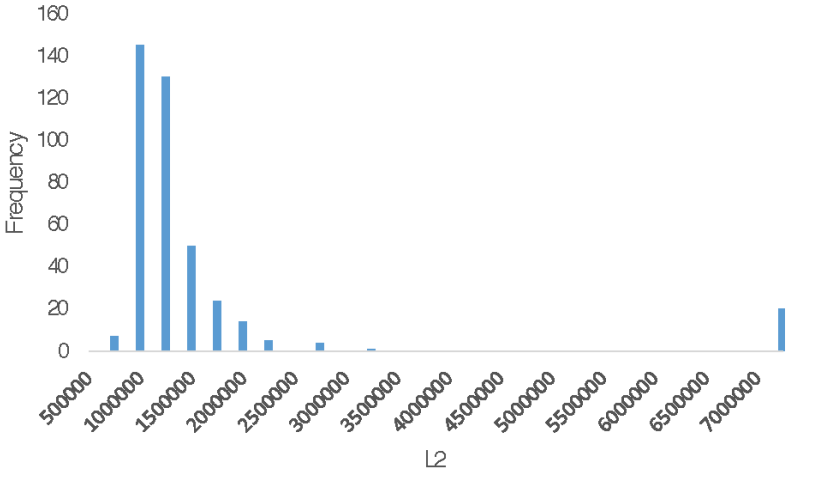

Figure 8. Distribution of fitness values of 400 runs of one particular parameter setting.

We now use the Artificial Anasazi model to demonstrate the 3

approaches. First, we determine the parameter space. We will evaluate 5 parameters. The parameters with the ranges and increments of those parameters are defined as follows:

```
"fertility" [0 0.001 0.2]
"death-age" [26 1 40]
"harvest-variance" [0 0.01 1]
"fertility-ends-age" [26 1 36]
"harvest-adjustment" [0 0.01 1]
```

This leads to 338,316,165 different parameter combinations. In our demonstration, we will run, for each of the three approaches, the model 100,000 times and show the parameter estimation and fit with data. We use the same number of runs for each approach in order to compare the results with the same amount of runs.

For the *Calibration*, we will calculate for each parameter setting we evaluate the median of the squared differences, L2, of 100 runs. Since we will do 100,000 runs, we will explore 1000 different parameter settings. We run the genetic algorithm in BehaviorSearch to find the best fit. Note that if we rerun the genetic algorithm, or run it longer, we may find different solutions. The best fit is conditional to the number of model runs we use in the search.

For *Pattern-Oriented Modeling*, we look at the maximum difference between the simulated and actual data and only accept parameter settings that will not have a bigger difference of 100 households between the data and the simulation for all 100 runs. We only look at the simulation until the year 1300 since we know that all model variations will not project a complete abandonment of the valley. We looked at a random sample of 1000 parameter settings and found only 1 parameter settings that met the criteria. To implement this in BehaviorSearch, we use Random Search with 100 replicated per Fixed Sampling in the tab "Search Objective."

For the *Approximate Bayesian Computation (ABC)*, we run the model 100,000 times each with different parameter settings and calculate the squared difference for each run. We then select the 100 best runs and calculate the parameter distributions. Figure

9 shows the best run. To implement this in BehaviorSearch, we use Random Search with 1 replicate per Fixed Sampling in the tab "Search Objective."

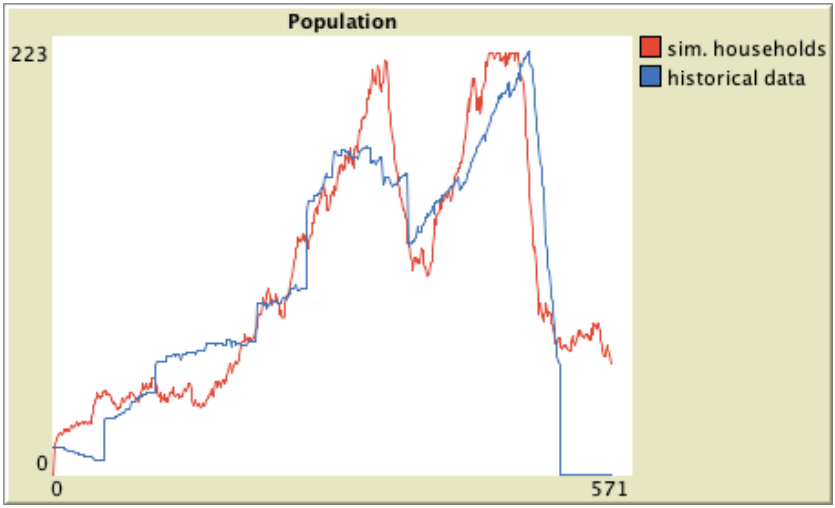

Figure 9. Best fit with data among 100,000 runs with different parameter settings.

In Figures 10-14, we see the parameter distributions of the *Approximate Bayesian Computation (ABC)*, and the optimal parameter settings for *Calibration* and *Pattern-Oriented Modeling*. We see that different approaches lead to somewhat different solutions. The harvest-adjustment rate has a narrow window of possible values in line with the analysis published in Janssen (2009). The calibration solution is different for Harvest variance, probably due to the larger number of runs used here (=100) compared to Janssen (2009), which was 15 runs as used by Axtell et al. (2002).

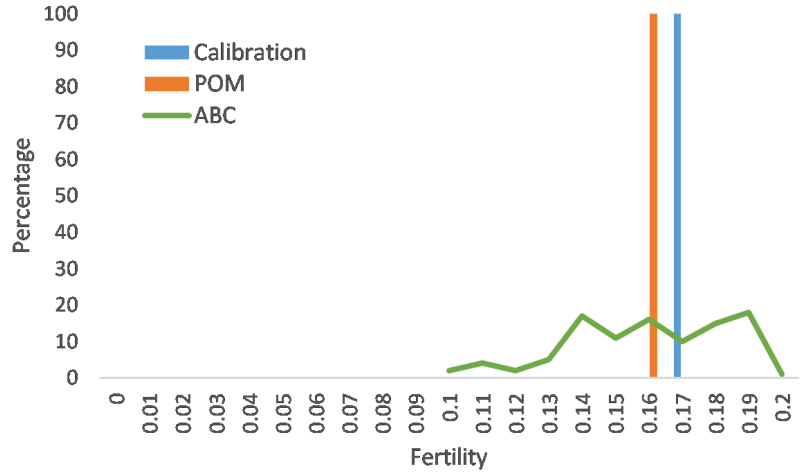

Figure 10. Parameter estimates for fertility using different methods.

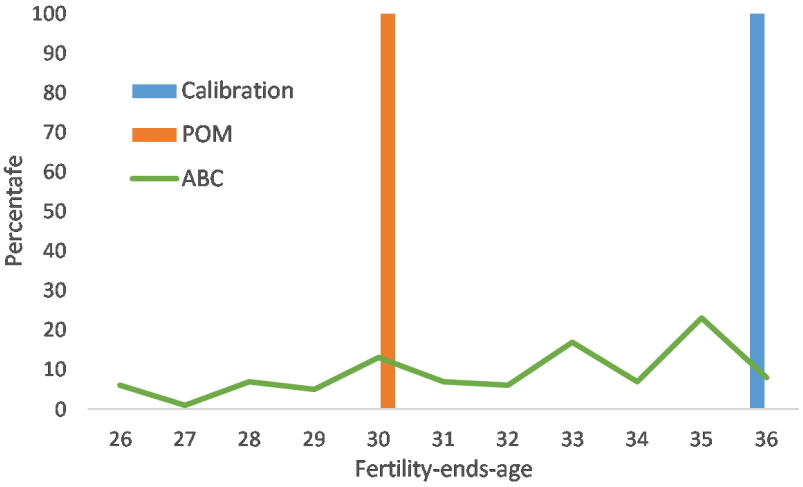

Figure 11. Parameter estimates for fertility-ends-age using different methods.

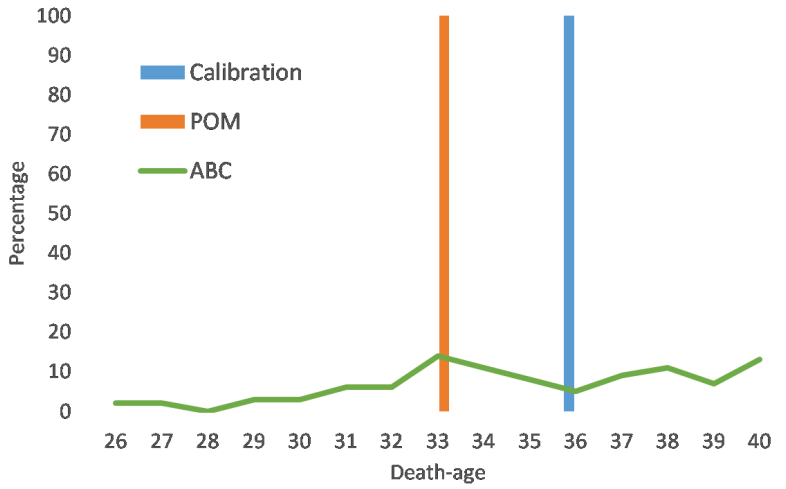

Figure 12. Parameter estimates for death-age using different methods.

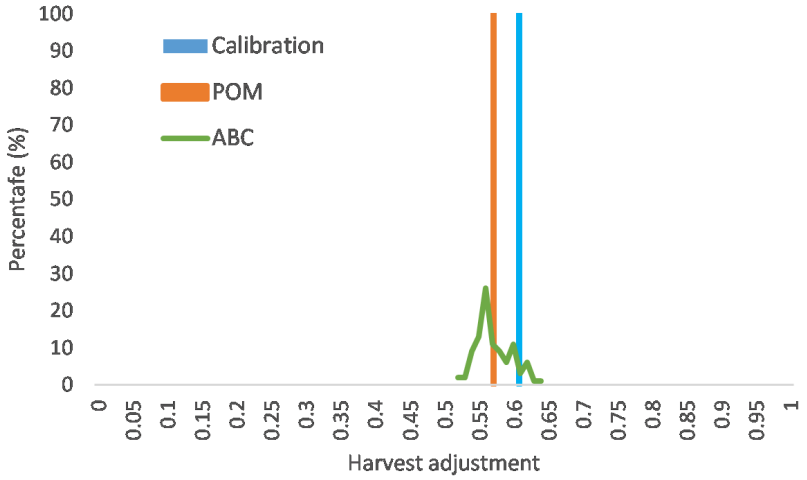

Figure 13. Parameter estimates for harvest-adjustment using different methods.

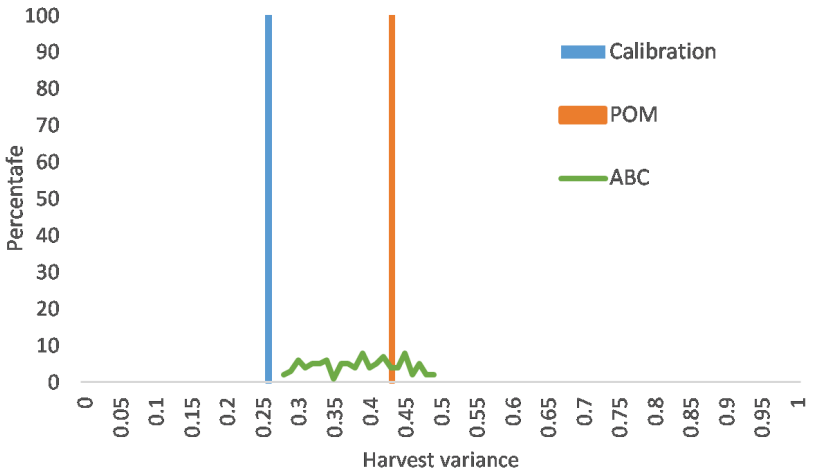

Figure 14. Parameter estimates for harvest-variance using different methods.

The results demonstrate that different parameter estimation methods may get different findings and insights. As such, one may use different approaches in a study to see the variance of possible outcomes.

Summary

This chapter provides a brief tutorial of BehaviorSearch, a NetLogo tool, to search the parameter space of a NetLogo model to find parameter settings that optimize a certain objective function. We can define an objective function as the difference between the simulated and empirical data and use BehaviorSearch to find those parameter values that best explain the observations. There are, however, different approaches to approach the evaluation of the model fit with the data. We used the well-known Artificial Anasazi model to demonstrate classic *Calibration*, *Pattern-Oriented Modeling*, and *Approximate Bayesian Computation*.

References

Axtell Robert L., Joshua M. Epstein, Jeffrey S. Dean, George J. Gumerman, Alan C. Swedlund, Jason Harburger, Shubha Chakravarty, Ross Hammond, Jon Parker, and Miles Parker (2002)

Population Growth and Collapse in a Multi-Agent Model of the Kayenta Anasazi in Long House Valley. *Proceedings of the National Academy of Sciences* 99(3): 7275-7279.

Beaumont, Mark A. (2010) Approximate Bayesian computation in evolution and ecology. *Annual Review of Ecology, Evolution and Systematics* 41: 379–406.

Grimm, Volker, Eloy Revilla, Uta Berger, Florian Jeltsch, Wolf M. Mooij, Steven F. Railsback, Hans-Hermann Thulke, Jacob Weiner, Thorsten Wiegand, Donald L. DeAngelis (2005) Pattern-oriented modeling of agent-based complex systems: lessons from ecology. *Science* 310: 987–991.

Janssen, Marco A. (2009). Understanding Artificial Anasazi. *Journal of Artificial Societies and Social Simulation* 12(4)13 http://jasss.soc.surrey.ac.uk/12/4/13.html.

Van der Vaart, Elske, Alice S.A. Johnston, Richard M. Sibly (2016) Predicting how many animals will be where: How to build, calibrate and evaluate individual-based models, *Ecological Modeling* 326: 113-123.

PART IV

ECOSYSTEM MANAGEMENT

In this part of the book, we will develop a series of basic ecological models of populations who search for food, reproduce and die, and explore how interventions impact the population size. We also use some more advanced ecological models, implemented as difference equations, to combine systems dynamics models and agent-based models, and perform an analysis to identify bifurcations.

CHAPTER 14

Population Ecology

Key Concepts

In this chapter, we will:

- introduce carrying capacity,
- present a model where agents are born and die,
- demonstrate factors that restrict the size of the population in a landscape,
- show phase diagrams to present model results.

How many people can our planet sustain? The answer depends on the amount of food produced, the energy available, the rate at which people consume resources, and how resources are shared. In this chapter, we will discuss the use of models to address questions on the carrying capacity of the environment. We present a model where agents interact with their environment. The agents need to have resources to survive. They will search the environment for resources and collect them. The amount of resources available is limited and can be regenerated at a modest pace. If the agents consume more resources than the environment

can produce, there might be a problem for the population in the long term.

Although the model presented in this chapter is somewhat abstract, it addresses several broader questions on sustainability.

Basic Model

Let's start with the basic model we will use in this chapter. The model consists of an environment and an agent who lives in this environment. The model used in this chapter can be downloaded here.

Environment

Given below is a landscape of N×M cells. The edges are connected to derive a torus – donut-shaped – environment (Figure 1).

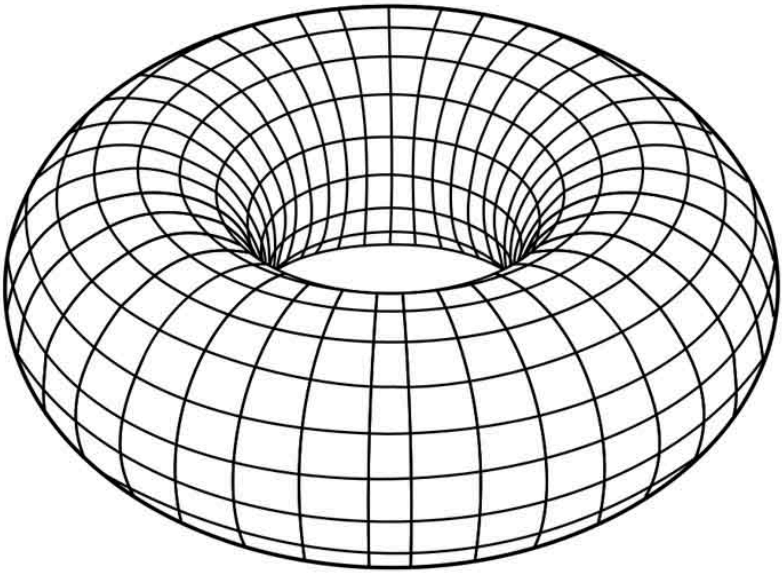

Figure 1. Torus Representation.

When we put agents on the landscape, they will not experience

any edges. Each cell can contain a resource unit. This resource unit provides an amount of energy *R* to an agent if the agent collects it. If an agent has collected the resource unit of a cell, a new resource unit can appear on the cell with a probability p_r.

To implement this in NetLogo, we will look at each patch to determine whether there are resources on the cell. If there are no resources on the cell then there is a chance prob that new resources will be added to the cell. In a simple model, the only options available are resources on the cell or no resources on the cell. This can be implemented as below. Figure 2 shows a screenshot of such a landscape.

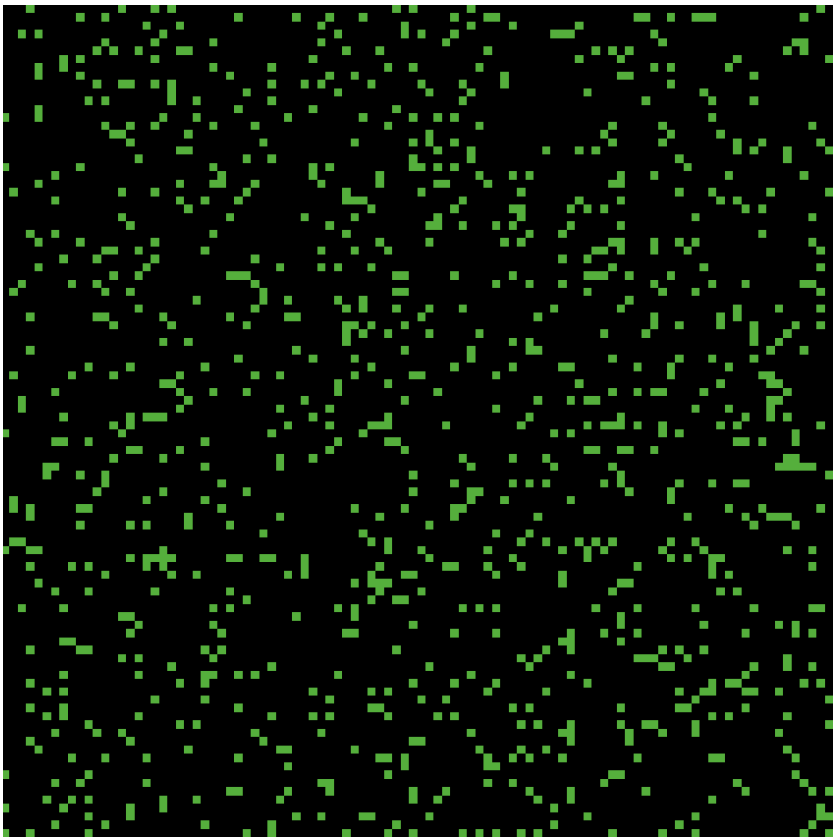

Figure 2. Screenshot of the landscape.

```
if pcolor = black [
  if random-float 1 < prob
  [
    set pcolor green
  ]
]
```

Agents

Agents need the energy to survive. Each time step, they use *m* units of energy. This is their metabolism. They can derive energy from collecting resource units from the landscape. Agents move

randomly around, and if they are on a cell with a resource unit, they collect this resource unit. If agents have run out of energy, they will die. On the other hand, if their energy level reaches more than *bt*, the birth threshold, the agent will split into two agents. This cloning is a simplistic way of simulating reproduction.

We will now discuss a NetLogo program that simulates the model described above. First, we have to define that agents have the attribute energy. This is the only turtle attribute that we need to include. Of course, the turtles already contain a number of default attributes such as location, color, and heading.

```
turtles-own [energy]
```

To initialize the model, we have to define the initial level of energy the agents have. In this example, we assume that at the start, all agents have 10 units of energy. We put the agents randomly on the landscape and represent them as white dots.

We also need to initialize the environment. For each patch, we flip a coin to define whether the patch has a resource unit or not. If the patch has a resource unit, we define the color of the patch green; otherwise, it remains black. Note that we used a standard attribute of the patch to define whether a resource unit is available, and so we will not have to define a new attribute for patches. The initial density of resources is assumed to be equal to initial-density, a value determined by a slider in the interface.

```
to setup
  ca
  set-default-shape turtles "dot"
  create-turtles number [
    set energy 10
    set color white
    setxy random-xcor random-ycor
  ]
  ask patches [
    if random-float 1 < initial-density
      [set pcolor green]]
```

```
end
```

Once we click on *go* the agents will make a number of decisions within each tick. The first problem we face is in which order decisions need to be made. Will an agent eat first and then move? Does each agent perform the same actions in the same order? In which order do we update the agents? Will some agents get priority? These decisions on the order in which decisions are made can have important consequences on the outcomes of the model. In our simple model, it will not have a huge impact. We assume that the order of the actions is movement, eating, metabolism, reproduction, and dying.

First, all agents will move before the next action. You see that the agents change their direction randomly before taking one step forward. To model the random type of movement we change the direction of the agents each time step. The heading of the agent is changed by moving it to the right. The number of degrees it moves to the right is a random number between 0 and 45. Then the agent moves to the left again, also at a randomly drawn number between 0 and 45. On average the heading of the agent is directed straight forward, but most agents will change its heading a small amount to the right or left. With a small change, the agent will change it's heading by several degrees in one step. After the direction is determined, the agent will make one step forward using fd 1.

```
ask turtles [
  rt random 45
  lt random 45
  fd 1]
```

If they are on a patch that is colored green (thus has a resource), the agent will collect the resource unit, which will generate penergy (slider value) units of energy. It is possible that more than one agent is on a cell. In that case, the agent who has drawn first to eat will collect the resource unit and put the color of the cell equal to black.

```
ask turtles [
  if (pcolor = green) [
    set pcolor black
    set energy energy + penergy]]
```

Then the energy level of all agents is reduced with the amount of metabolism (slider value).

```
ask turtles [set energy energy - metabolism]
```

If the energy level is more than the birth-threshold (slider value), we first half the agent's energy level before we generate a copy. Hatch means that the agent is cloned. If we don't half the energy first, we will get two agents with more energy than the birth-threshold. In such a situation, the system would generate energy by producing offspring, which is not possible.

```
ask turtles [
  if energy > birth-threshold [
    set energy (energy / 2)
    hatch 1]]
```

Finally, if an agent has a negative unit of energy, the agent will die and is removed from the system.

```
ask turtles [if energy < 0 [die]]
```

We also need to update the resource units on the patches every time step. If a patch has no resource unit, we assume that with a probability replenishment-rate (slider-value) a new resource unit will be added to the patch. The NetLogo model for this was discussed above when we discussed the environment.

Now we can simulate the model in NetLogo. In Figures 3 and 4, you see some typical results of the model. You will see that the population level depends critically on the level of the parameters, which can be changed by the sliders. If metabolism is high compared to energy per resource unit, the population is more quickly to die out. In other situations, we see that the population

reaches a somewhat stable level of resources, or has some periodic fluctuations. Why do we get fluctuations of the population size? The agents eat up the resources quickly and produce a lot of offspring. The increasing population causes a scarcity of resource units. Due to the scarcity, agents cannot get enough energy to maintain their metabolism, and more agents will die than are born. When the population size drops, more resources become available for those who survived, etc.

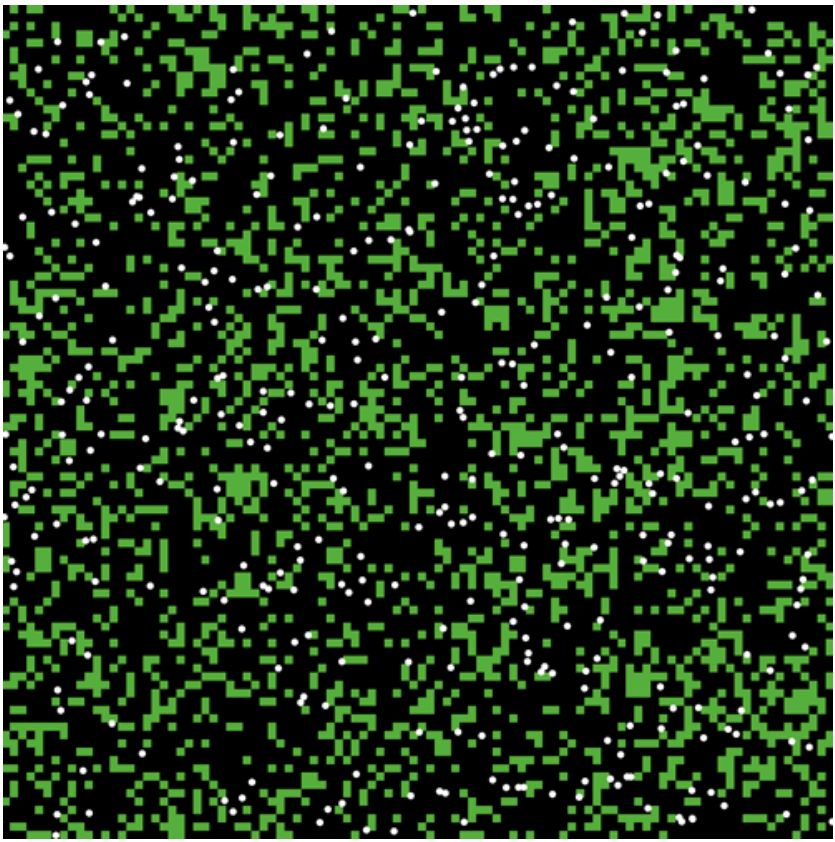

Figure 3. Screenshot of a typical run of the model.

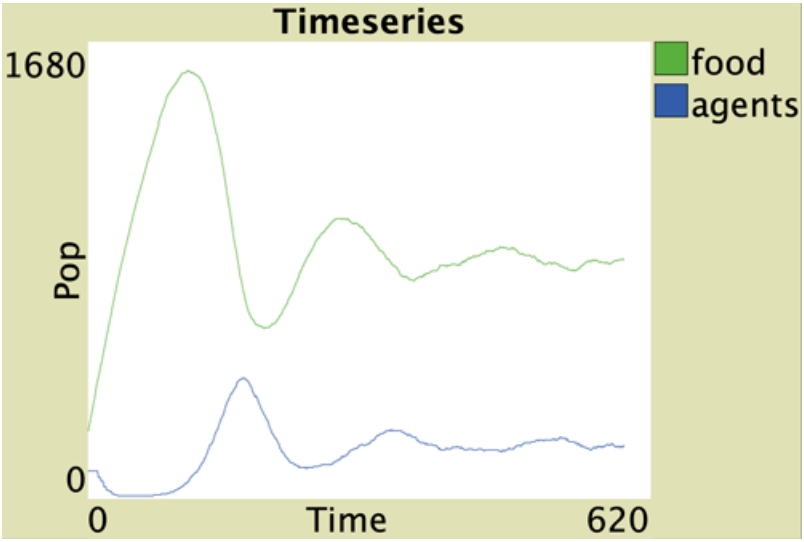

Figure 4. Example of outcomes of the population level (blue) and resource (green) over time.

How many agents can survive in this landscape? The default parameters of the agents are

```
metabolism = 1
birthrate = 20
foodenergy = 3
```

The environment is 100 x 100 cells, thus 10,000 cells. Suppose the environment produces foodenergy * replenishment-rate * 10,000 cells, then the environment produces 300 units of energy is the replenishment rate is equal to 0.01. This could feed 300 agents. However, agents will have to move to another cell to find food. Agents move randomly, so it will not be the most efficient exploration. At least we can expect that the total number of agents that survive over the longer term will be lower than 300 agents. When we run the model, we see that the population stabilizes around 200 agents. For each increase in the replenishment rate, we can expect a linear increase in the population.

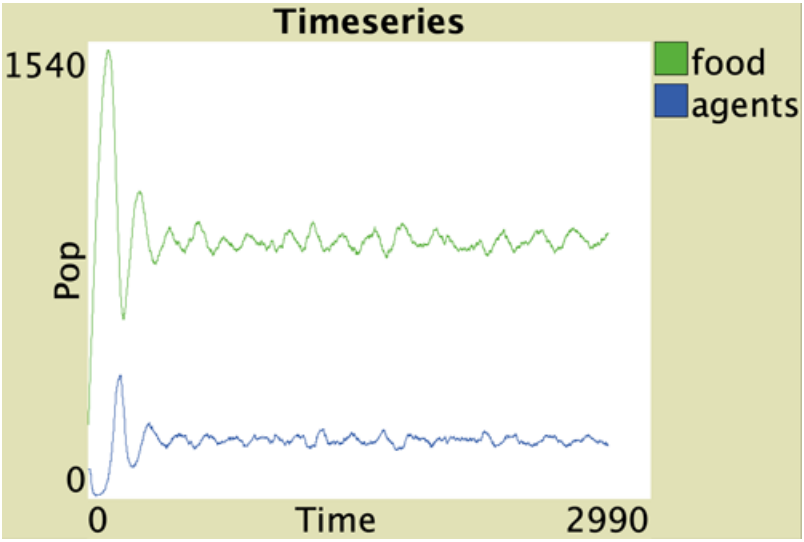

Figure 5. Resource size and population size using a replenishment rate equal to 0.01.

Density Dependent Resource Growth

We assumed in the previous section that the resource units will always recover from an empty cell using a fixed replenishment rate. This is very unlikely with many resources. The more trees in an empty spot in a neighborhood, the more likely that seedlings will fall on this spot, and a new tree will be generated. If there are no trees around, no new trees will regenerate. This is called density-dependent growth, and it is a common mechanism in many ecosystems.

For the whole environment, we can now expect a logistic growth. In fact, the model above is a spatially explicit version of the famous logistic growth function. Suppose the resource size is defined by X[t] for time step t. Then the resource size at time step t + 1 is defined as

X[t+1] = X[t] + r * X[t] * (1 – X[t] / K)

Where r is the regrowth rate and K the carrying capacity. In our environment above, K is 10,000, namely 10,000 cells, where each

cell can only be occupied by one agent simultaneously. The highest regrowth rate will happen when half of the resource is available, meaning X[t]= 5,000 for K = 10,000. If r is equal to 0.1, the resource grows over time, as in Figure 6. This is the classic S figure of the logistic growth function.

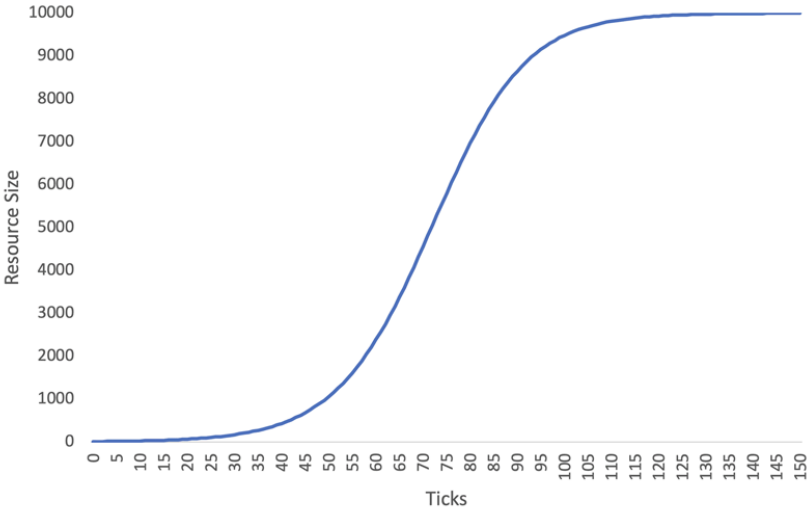

Figure 6. Resource level of logistic growth model with r = 0.1 and K = 10,000.

In a spatial model, the regrowth rate not only depends on the value of r, but also on how the resource is distributed spatially. To model this in NetLogo we may initially think about the following implementation:

```
ask patches [
  if pcolor = black [
    if random-float 1 < replenishment-rate *
      (count neighbors with [pcolor = green])]
      [set pcolor green]
  ]
]
```

This means that for every patch with no resource, there is a

probability that it will regenerate. This probability will be higher the more neighboring cells that have resource units.

However, this implementation is often affected by a typical error people make in agent-based models. The problem is the following: All cells are updated randomly, one by one. If you have an empty cell, and this cell becomes a cell with a resource unit, this cell is no longer seen as empty by the next cell to be updated. This can lead to strange results since the order in which cells are updated affects the results. It is theoretically possible that an empty landscape with just one cell with a resource unit could lead to a complete recovery in one time-step. This is an unrealistic representation.

So how to implement the regeneration of the cells where we only take into account the state of the cells at the start of the time step? We need to add an additional attribute to the patch that remembers the old state of the cell. Then we can correct the model in the following way.

```
ask patches [set oldstate pcolor]

ask patches [
  if oldstate = black [
    if random-float 1 < replenishment-rate *
      (count neighbors with [oldstate = green])
    [set pcolor green set penergy food-energy]]
  ]
]
```

With a replenishment rate equal to 0.01, we get the dynamics we observe in Figure 7. The resource and population numbers do not stabilize and keep going up and down. When there is a lot of food, the agents start to consume a lot and produce a lot of offspring. This leads the resource levels to go down, making it harder for the agents to find food. Since their metabolism needs energy every time step, some agents will die. As a result, the population size falls, and fewer resources are eaten. This, in turn, leads to the rise of the resource.

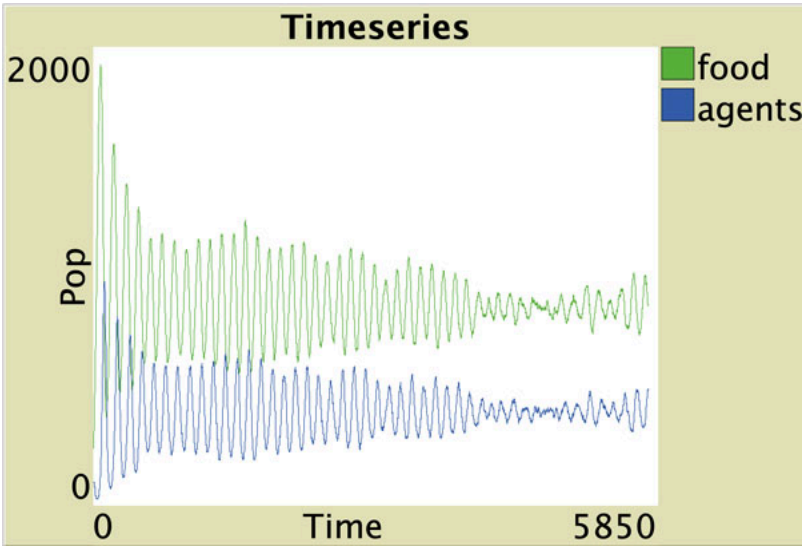

Figure 7. Population and resource size for density-dependent regrowth of resource with regrowth rate equal to 0.01. (Note that food is resource size divided by 4 to have the numbers at a similar scale.)

We can portray the dynamics of the population and resource in a so-called phase diagram (Figure 8). This shows the cyclical dynamics of the two variables leading the system to circle around in a confined area of the phase diagram but never settle into equilibrium.

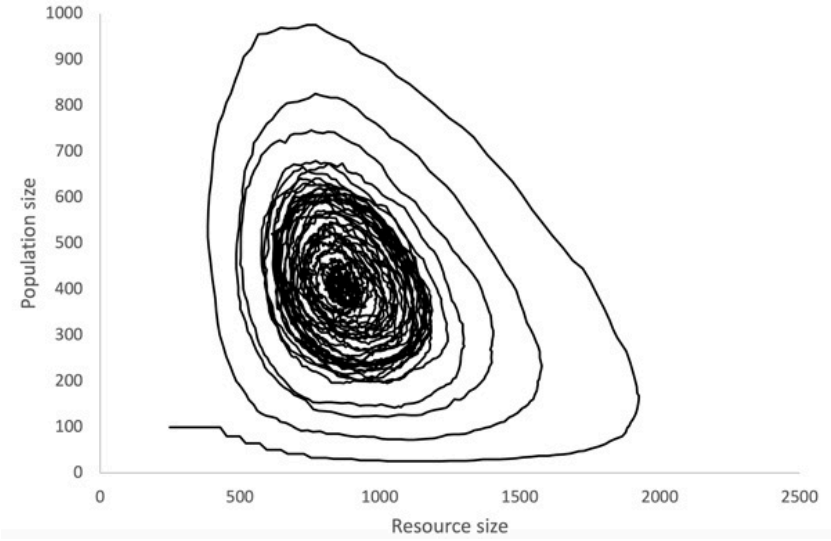

Figure 8. Phase diagram of population and resource with regrowth rate of 0.01 (generated in Excel with results from Figure 7).

When we increase the regrowth rate to 0.1, the population level circles between 2500 and 5500, but never stabilizes (Figure 9). The population does not die out. When the resource is low, agents die out, and the resource recovers. However, most species do not reproduce by cloning, and we will explore the consequences of having two parents in the next section.

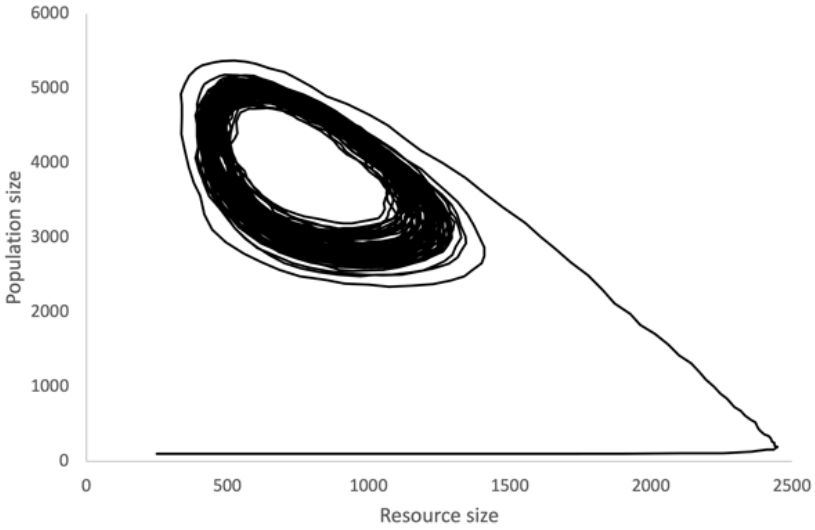

Figure 9. Phase diagram of population and resource with regrowth rate of 0.1.

The Effects of Mating

Suppose agents can only reproduce when two agents meet who have energy beyond the minimum level of energy. Such a rule will imply that not all agents with more energy than the birth-threshold will get offspring since each potential parent needs to find a suitable mate.

To implement this, an agent who has the minimum energy required for producing offspring needs to look around and check whether there is an agent nearby with sufficient energy. The primitive in-radius gives the set of agents that are found within a radius of a defined size. Thus turtles in-radius search-radius with [energy > birth-threshold] is the set of agents in the radius of size search-radius who have more energy than the minimum amount of energy needed to produce offspring.

We only want to move on if there are any agents that meet this condition. Thus before moving on, we use primitive any?, which checks whether there are agents within the agent set, and thus

```
if any? turtles in-radius search-radius with [energy > birt
```

Now there is another problem. The agent who is looking for a mate is included in the agent-set since it is an agent within the search radius. Therefore, we have to find a way to exclude the agent who is searching for a mate. Consequently, we check what the identity is for the agent looking for a mate and exclude this from the list of agents that become a mate

```
let agentwho who
if any? turtles in-radius search-radius with [energy > birt
```

So, at this point, we have a set of candidate mates. The agents are not very picky. We assume they skip speed dating and just accept any mate that meets the minimum conditions. Hence to select the mate, we take one agent of the set of candidates. The number of candidates might be one, and then that one is chosen. But if the set is larger than one, one agent is randomly selected.

```
let mate one-of turtles in-radius search-radius with [energ
```

We assume that the initial energy of the child is 10% of the combined energy of the two parents. Each parent contributes an equal share to the energy of the offspring. This is implemented as:

```
let childenergy (energy + [energy] of mate ) / 10
set energy energy - 0.5 * childenergy
ask mate [set energy energy - 0.5 * childenergy]
  [ hatch 1 [set energy childenergy] ]
```

When we run the model with a regrowth rate equal to 0.01 and a search-radius of 1, we get the phase diagram of Figure 10. The amplitude of the cycle is a bit larger than Figure 8, without mating. What is the reason for this?

Figure 10. Phase diagram for population and resource size with mating.

Summary

This chapter presents a basic population model. Such a model could be described by difference or differential equations, assuming a large enough population. We could also use visualizations of agent-based model results like used more commonly for difference equation models, such as phase diagrams. When we introduced mating, we included an addition that would be more suitable for agent-based model implementation. Hence this chapter demonstrates that models typically studied with difference equations could also be implemented by agent-based models, and due to the stochastic nature of agent-based models derive similarly, but not the same, results.

References

Durrett, Richard and Simon Levin (1994) The Importance of Being Discrete (and Spatial), *Theoretical Population Biology* 46(3): 363-394.

CHAPTER 15

Governing the Commons

Key Concepts

In this chapter, you will

- be introduced to the tragedy of the commons,
- demonstrate the tragedy in a NetLogo model,
- be shown that agents can self-govern their shared resources.

Resources are appropriated, and when groups of people share resources, it can be a challenge to avoid overharvesting of the shared resources. For example, consider fishers fishing the oceans, farmers pumping up groundwater, oil companies drilling for oil, or movie watchers using the limited bandwidth of the community internet connection. In 1968 biologist Garrett Hardin wrote a famous essay in the journal Science titled "The Tragedy of the Commons." He used the metaphor of sheepherders sharing a meadow. While each herder will benefit as an individual from adding an extra sheep to the pasture, as a group, they will bear the costs of the additional grazing and especially when it creates a situation in which grazing occurs faster than the resource can

be regenerated. All herders share the cost of overgrazing, but the benefit of the extra sheep goes to the sheep's sole owner.

To avoid overharvesting of the resource, Hardin argued that there is a need for strict governmental regulation or privatization of the resources. Without these interventions, the herders would inevitably overharvest. Hardin's essay was highly influential among emerging environmentalists who were becoming increasingly concerned with the many human-caused environmental problems. They called the government to action, and new regulations were written to reduce pollution and avoid overharvesting.

Since Hardin's essay, increasing awareness also emerged that a tragedy is not the only possible outcome when people share a common resource. There are many examples of long-lasting communities that have maintained their common resources effectively. Since the 1980s, there has been an increasing interdisciplinary effort to debunk the simplistic view of the tragedy of the commons. Elinor Ostrom and others showed through a comparative analysis of many case studies that humans can self-govern their common resources. In her classic study "Governing the Commons," Elinor Ostrom defines eight principles that enhance the likelihood of self-governance, including effective monitoring and strict boundaries of the resource.

In this chapter, we show through some simple models what happens when agents consume shared resources and are greedy, or when they imitate other successful agents and what happens when monitoring and sanctioning is included.

Modeling the Tragedy

We present a simple model where agents harvest from a shared resource without restrictions. We will see that this leads to the overharvesting of the shared resource. There is a number of tweaks to the model, which introduces some new primitives again. We used the Wealth Distribution in the Social Science folder of NetLogo by Uri Wilensky as the starting point. Our model version that covers this section and the next can be downloaded here.

First, we model the resource. Not every cell has the same carrying capacity. First, we set a percentage percent-best-land of the cells at the maximum carrying capacity. This is the best land in the landscape.

```
ask patches
[
  set max-resource-here 0
  if (random-float 100.0) <= percent-best-land
  [
    set max-resource-here max-resource
    set resource-here max-resource-here
  ]
```

This leads to a randomly distributed number of cells that have the maximum resource level as shown in Figure 1.

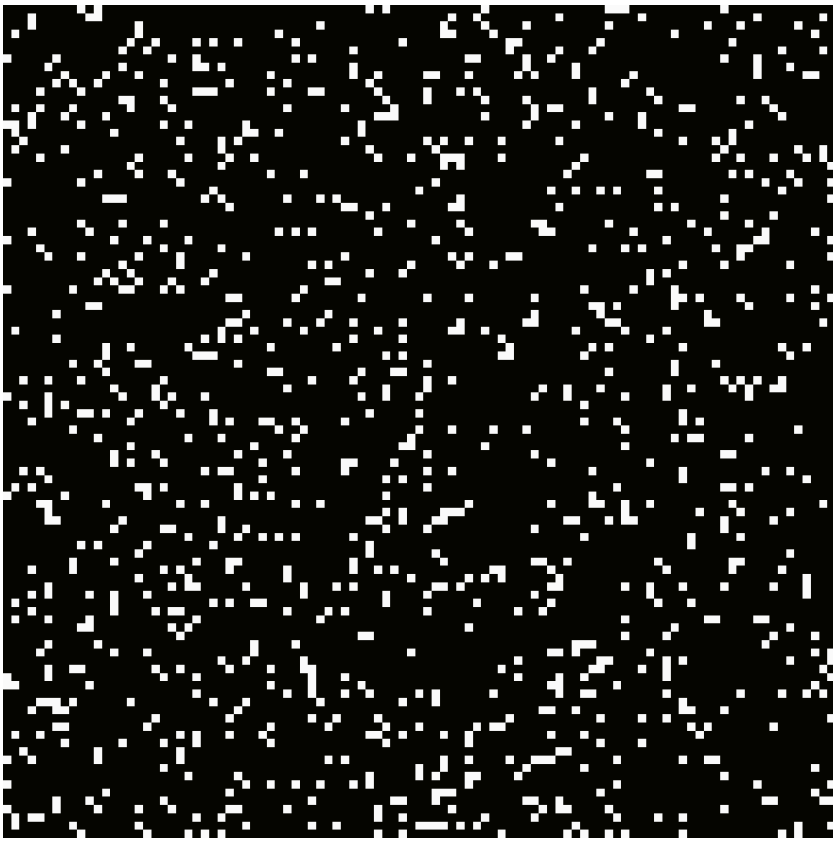

Figure 1. Distribution of patches with resources at the maximum level (white patches).

Subsequently, we use the primitive diffuse to spread resources to neighboring patches. Diffuse resource-here 0.25 means that 25% of the resource level of a patch is distributed to the 8 surrounding patches. This is done for 5 iterations for patches that have resources on their patch in the code below. This leads to resource distribution, as in Figure 2.

```
repeat 5
  [ ask patches with [max-resource-here != 0]
      [ set resource-here max-resource-here ]
```

```
      diffuse resource-here 0.25 ]
```

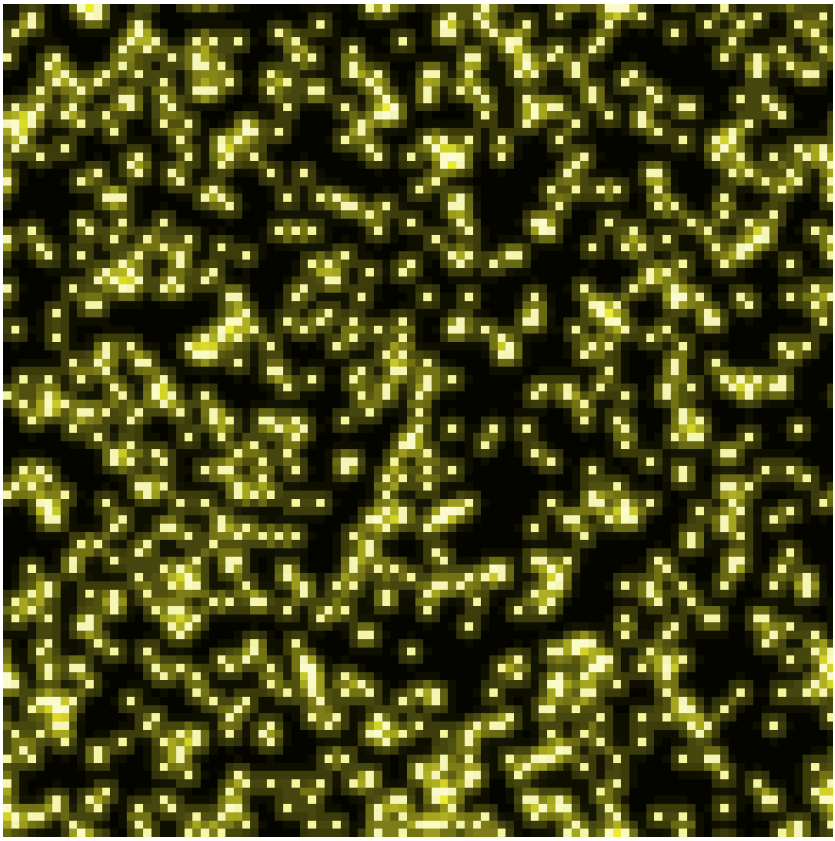

Figure 2. Distribution of resources on patches.

```
repeat 10
[
  diffuse resource-here 0.25
]
```

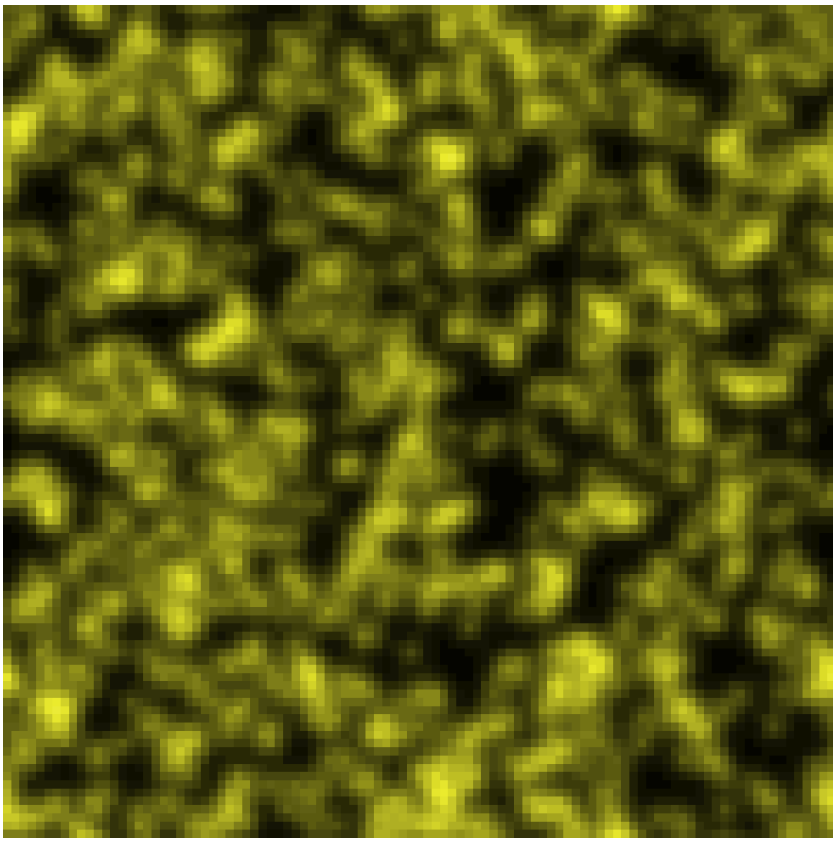

Figure 3. Final distribution of resources, equivalent to carrying capacities of the patches. Lighter colors refer to higher levels of the resource.

An agent looks to the north, east, west, and south patch. It will look a number of patches ahead according to its vision and determine the resource level of the patch with the highest resource level for each direction. The agent then determines which of the neighboring directions has the highest level of the resource. If there is more than one neighbor with the highest resource level, one is drawn at random. The agent then turns toward the best neighboring patch and makes a step forward.

Each tick the agent will harvest as long as the patch has more

than one unit of the resource available. The resource will regrow using a logistic equation using a regrowth rate of 1% per tick. Figure 4 shows the results when agents take resources whenever they are available. These greedy agents lead to a collapse of the resource.

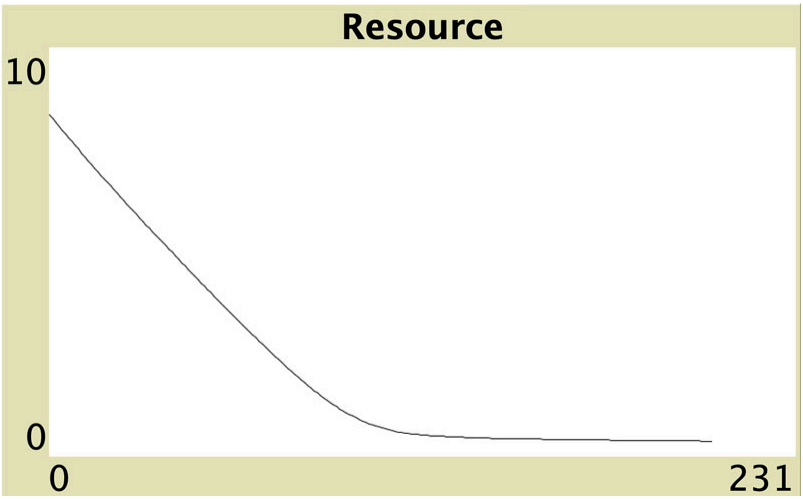

Figure 4. Average resource size over time for greedy agents.

If we include a heuristic that agents will not harvest if the resource is below a certain level, we may derive a more sustainable outcome. For example, what will happen if we assume that agents will not harvest a resource when it drops below 1 unit or less than 50% of that cell's carrying capacity. Since the resource grows according to a logistic growth function, the maximum growth rate happens with a resource level of 50% of the carrying capacity. Hence if agents are harvested at 50% of the carrying capacity, they will receive a maximum return in the longer term. This outcome is often called the maximum sustainable yield. Suppose we have the logistic growth equation for resource X:

$$X[t+1] = X[t] + r * X[t] * (1 - X[t] / K) - H[t]$$

Where H[t] is the harvest at time step t, r is the regrowth rate, and K is the carrying capacity. In the long term, the sustainable solution

is to harvest each time step the same amount as the resource generates. This means that

$$H[t] = r * X[t] * (1 - X[t] / K)$$

If X is equal to 0, meaning a collapse of the resource, the harvest level is 0. The other equilibrium is when X is equal to 0.5 * K. This is derived by taking the derivative of $rX(1 - X/K)$ and set it equal to 0.

Figure 5 shows the result is the heuristic below 50% of the carrying capacity. We see that the resource is not collapsing but stays at a steady level above 50% of the system's carrying capacity since agents do not harvest the resource below 50% of each patch's carrying capacity.

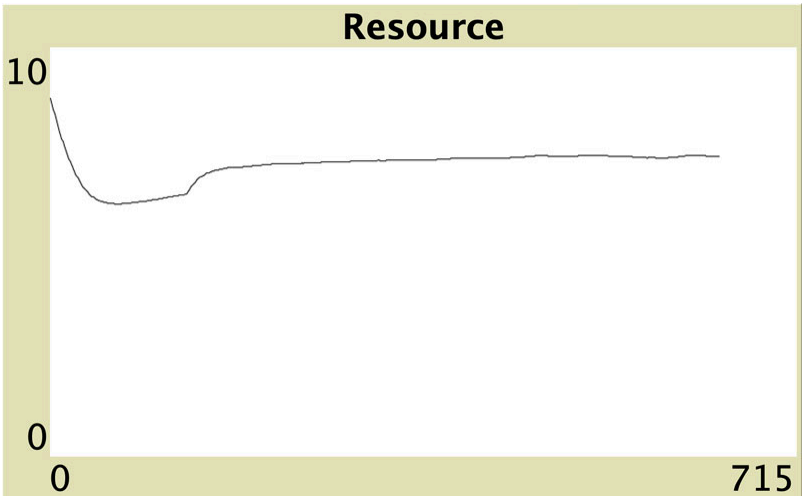

Figure 5. Average resource size over time for the norm-obeying agents.

What if agents can imitate the behavior of others? We assume that agents all have different thresholds and imitate norms used by other agents who have greater wealth than themselves. Wealth is defined as the discounted sum of earnings of the past. Each tick, past earnings are multiplied by 0.95, which means a discount rate of 5% (a reduction of 5%). This means that recent earnings

have a higher weight than earnings far in the past. What is the consequence of this success-biased imitation?

If agents imitate the threshold of others perfectly, and all agents start with the same threshold of 50% of the carrying capacity, we will not see any differences compared to Figure 5. Although some agents may earn more than others, there is no diversity of traits. However, if agents vary in their initial norms between 0% and 100%, we see that the average threshold value goes down (Figure 6). This means that agents are more eager to harvest resources from patches with a lower amount of resources. The reason for this is that agents who also harvested patches with a lower amount of resources derive more wealth than those who start looking for new resources quicker. The imitation of those agents who collect more resources leads to a race to the bottom.

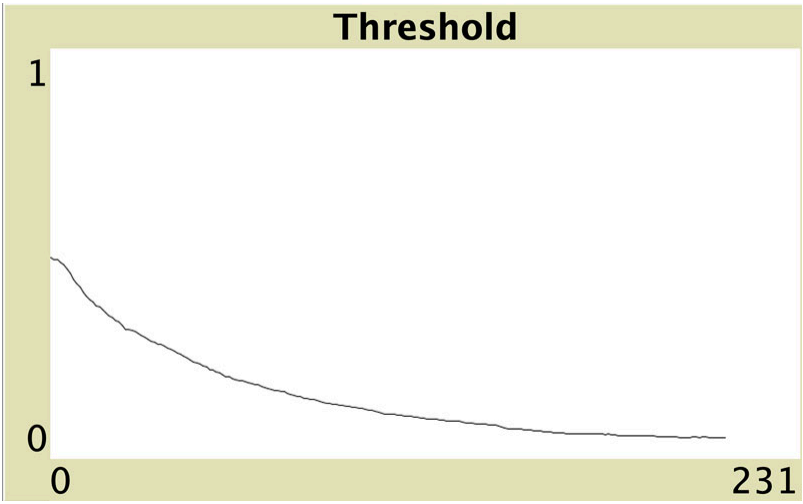

Figure 6. Average value of threshold-min-resource during a typical run.

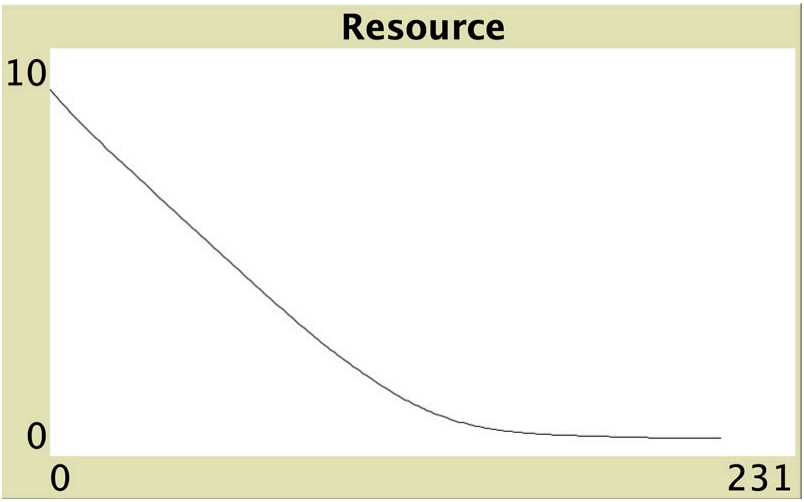

Figure 7. The average resource size during the same simulation as Figure 6.

Allowing for Self-Governance

We now introduce that agents may make mistakes in imitating the thresholds that others use. If they imitate, they imitate the threshold used by the agent they imitate, but we will add a noise term. All agents will start with a norm value of 50%. The mistake in imitation may lead some agents to start using a threshold above 50% and others below 50%. Over time we see that the threshold will drive towards 0%, and the resource will be depleted as shown in Figure 8.

How do we interpret this result? Agents are willing to obey a threshold of sustainable use of the resource but adjust their strategy if they see that others do better by not obeying the threshold. In the model, all agents have good intentions, yet just by imperfect copying, we get a crowding out of the moral norm of sustainable use of the resource. Hence good intentions, are not sufficient for sustainable use of the resource.

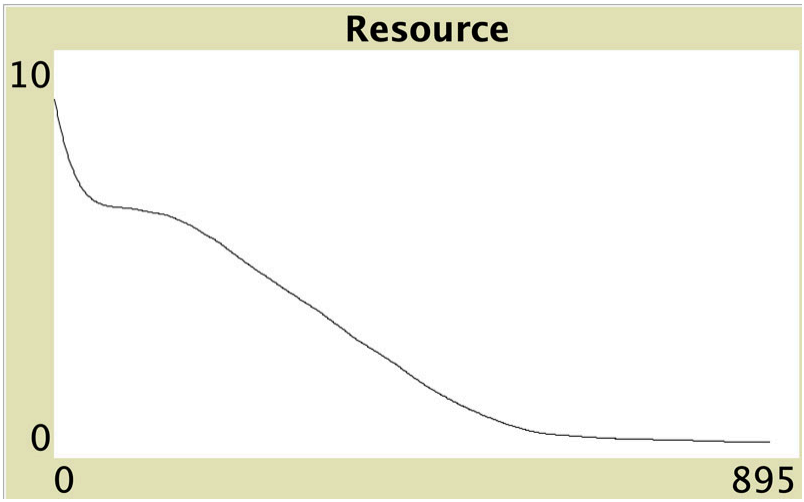

Figure 8. Average resource size over time for agents who imperfectly imitate better-performing agents.

The evolving thresholds can be seen as norms in a population. The example above showed that if agents just imitate thresholds of more successful agents, selfish behavior is rewarded, and the tragedy of the commons happens. So to allow more cooperative norms to evolve, we need to include enforcement.

We now include the option that agents can give each other a warning. For a small amount of cost, an agent can reduce the earning of another agent who is caught using a lower threshold. In the least, an agent can sanction those who harvest when they themselves would not harvest.

Each agent will check which agents in its vision has a lower harvestedlevel compared to the min-norm-resource of the initial agent. The variable harvestedlevel is calculated in harvest by dividing resource-here by max-resource-here. The primitive in-radius defines a radius of size radius around the agent.

```
let threshold min-norm-resource
let cheatingagents turtles in-radius radius with [harvested
if cheatingagents != nobody [
```

```
  set wealth wealth - count cheatingagent * costpunish
  ask cheatingagents [set wealth wealth - costpunished]
]
```

When there are other agents in the radius that use lower norms than itself, the agent will punish all those agents by reducing its wealth with costpunished, at the cost of costpunish to itself. Agents are not allowed to punish if their wealth is not more than costpunish. Hence an agent will have to harvest to be able to punish, and therefore the norm will evolve to very high values (which would mean agents rarely harvest from the resource). In the results below (Figures 9-11), we see that with a small radius (= 5), it is challenging to get cooperative norms evolving. When the radius is 10, cooperative norms evolve when the penalty is six times higher than the cost of punishment. But a very high penalty will not benefit society as it will lead to a loss of wealth, assuming the penalties are lost to society. Note that the results are obtained for running the model 2000 time steps for 100 times for each of the parameter combinations.

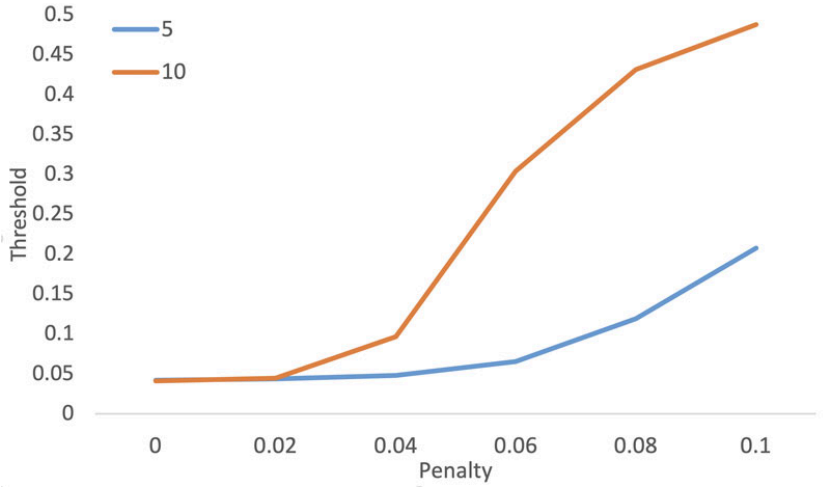

Figure 9. The average value of the threshold as a function of the penalty level cheaters experience for two radius levels.

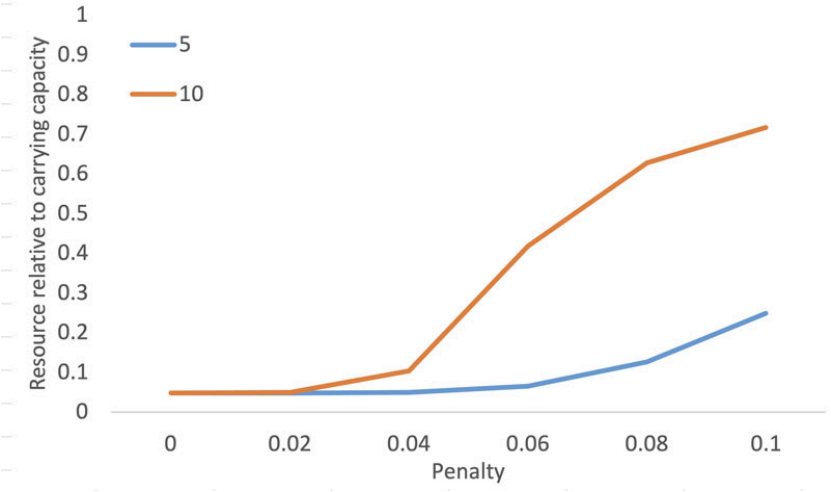

Figure 10. Average resource level relative to the carrying capacity as a function of the penalty level cheaters experience for two radius levels.

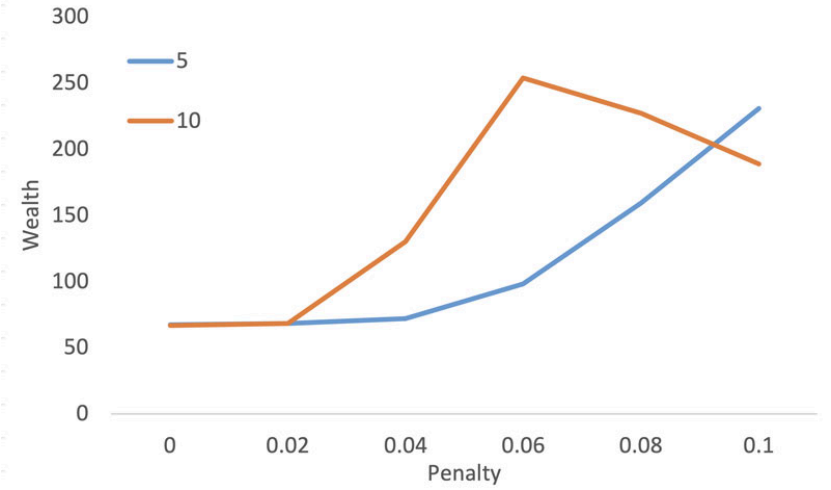

Figure 11. Average wealth per person as a function of the penalty level cheaters experience for two radius levels.

One of the challenges is to anchor the norm to a sustainable level.

The agents use informal norms and peer punishment. How do you feel about punishing your peers? Do you walk up to people who smoke in places they are not allowed to smoke, or if somebody throw trash on the street? We typically avoid peer punishment and create formal rules that are enforced by appointed enforcers. But enforcers do not have eyes everywhere. An extension of this model is to model appointed enforcers who're job is to enforce the rules. How many enforcers do you need to achieve sustainable outcomes? Can you rely on appointed enforcers only? Another extension is to implement graduated sanctioning. Agents who persist in breaking the rule may get higher penalties. First-time agents break a rule, get a warning, and get higher and higher sanctions over time. You may adjust the model to include those aspects and see whether you get a sustainable outcome.

Summary

In this chapter, we discussed the issue of managing shared resources. The traditional perspective, popularized by Garret Hardin, was that individuals compete for their resources in a rat-race to collect the benefits from the resource and share the costs of overharvesting with others. We also demonstrated a NetLogo model with an alternative perspective based on empirical observations of self-governed communities. Individuals enforce norms about keeping depleted resources off-limits by punishing those who do not follow the norms. This leads to sustainable outcomes of resource governance.

References

Anderies, John M. and Marco A. Janssen (2013) Sustaining the Commons, Center for Behavior, Institutions and the Environment, Arizona State University: free e-textbook at https://sustainingthecommons.org/

Bowles, Samuel (2008) Policies Designed for Self-Interested Citizens May Undermine "The Moral Sentiments": Evidence from Economic Experiments, *Science* 320: 1605-1609

Hardin, Garrett (1968) Tragedy of the Commons. *Science* 162: 1243-1248.

Ostrom, Elinor (1990) Governing the Commons. Cambridge University Press.

CHAPTER 16

Managing Rangelands

Key Concepts

In this chapter, you will

- explore models that are combinations of difference equations and an agent-based model,
- learn how to do resilience analysis with agent-based models,
- use optimization to find the best management strategies.

Rangelands are the semi-arid regions of the world that are too dry for reliable crop cultivation and hence used for livestock production of one form or another. They span the tropics and temperate zones, varying considerably in their vegetation and native fauna. The vegetation is characteristically a mixture of grasses, shrubs, and trees, ranging from pure grasslands to the woodland savannas of the subhumid tropics. Depending on the kind of rangeland, the welfare of the pastoralists who live in them is based on grazing animals (cattle and sheep), mixed feeders (browsers and grazers like camels and goats), or a combination of both. Fire has been an integral part of the environment of

rangelands, and the net effect has been to maintain rangelands in more open, grassy states than would be achieved in the absence of fire. Fire is not a disturbance in most rangelands; fire suppression is a disturbance.

In this chapter, we focus on a model of rangeland management in semi-arid Australia. We use this model to discuss the resilience of social-ecological systems, and illustrate how we can use models to find robust management strategies in systems with high levels of variability.

The models discussed in this chapter are based on research the author did in the early 2000s with colleagues of the Commonwealth Scientific and Industrial Research Organisation in Australia (Janssen et al., 2000, 2004; Anderies et al. 2002; McAllister et al., 2006) (Figure 1).

Figure 1. The author in Australian rangelands.

Resilience and Multiple Stable States

The concept of resilience in ecological systems was first introduced in 1973 by the Canadian ecologist C.S. "Buzz" Holling in order to describe the persistence of natural systems in the face of disturbances such as fires and pest outbreaks (Holling, 1973). A single system can have multiple types of states, for example, a lake can be either clear or green and murky, rangeland can have a mix of healthy tall grass and with a few trees and shrubs, or it can be covered with noxious weeds. If a system state is resilient, it remains in that state even if it is exposed to disturbances. If a system state loses it's resilience, for example, due to fire suppression, decades of nutrient loading in lakes, or overgrazing rangeland, it may not be able to recover from even a small disturbance, which would cause it to flip into an undesirable state.

The recognition that systems have multiple stable states leads to the concept of regime shift, which is a persistent change in the structure and function of a system, with substantive impacts on the suite of functions provided by these systems. In mathematical terms, the system experiences bifurcations, which means that in systems of differential equations, a small change of a parameter leads to a change in its system's behavior. Originally resilience analysis was performed using differential equations inspired with concepts from chaos theory and catastrophe theory.

We also find two stable configurations in rangeland systems: grass-dominated with periodic fire, and woody-dominated with no fire. The shift between attractors as grazing pressure increases may not happen smoothly. For example, the system remains in the grass-dominated attractor until the grass and shrub biomass, driven in part by grazing pressure, reach critical levels beyond which the dynamics of the rangeland change rapidly. The change in dynamics implies that the rangeland changes suddenly from being attracted to a grass-fire cycle to a system that is attracted to the shrub-dominated state. The change in vegetation itself may be quite slow, but the dynamics change immediately when the threshold is passed. This lack of rapid change in observed state

variables makes managing for such flips very difficult, especially when the system exhibits hysteresis. In such a case, reducing grazing pressure to the level which caused the initial flip is not sufficient to induce the system to return to the grass-dominated attractor. Instead, it must be reduced considerably below the critical level to cause the system to "flip" back into the attractor, where the dynamics will lead to a grass-dominated configuration. Thus, although recovery from a shrub-dominated state is possible, the time required is usually very long. On time scales relevant to management, the rangeland ecosystem is characterized by multiple stable attractors (Westoby et al. 1989).

Spatial Rangeland Model of One Property

In this section, we present a mean-field description of a rangeland system and subsequently implement it as an agent-based model to analyze the implications of non-uniform grazing (Janssen et al. 2002). The mean-field description is based on Anderies et al. (2002). With this model, Anderies et al. assessed the conditions under which the system flips from a healthy state to an unproductive shrub state. With the agent-based version, we will simulate individual sheep on a spatial lattice, and explore the consequences of different assumptions related to sheep's behavior, such as herding and the location of water points.

The model describes the interactions between perennial grass, shrubs, fire and commercial stock in a stylized way, based conceptually on the functioning of semi-arid woodlands and shrublands in western New South Wales in Australia. The grass plant consists of the crown, the root system, and the shoots, the above-ground grass portion of the plant. The biomass of grass shoots is denoted by s and follows a traditional logistic function. The crown promotes the growth of the shoots according to the tiller potential $c * a_c$ independent of grass biomass, and its interaction with above-ground biomass via the term $c * s$. Competition between woody shrubs and grass reduces grass growth. This is captured by the term $\alpha_{ws} * w^{\beta}$, where α_{ws} is a competition

coefficient, and where β (>1) leads to a growth reduction effect of woody shrubs that does not kick in until shrubs reach a relatively high density. Grass is removed by grazing pressure via the term $y_g * s$. Finally, grass biomass can be consumed by fire I, which has a general response function of form f().

$s[t+1] = s[t] + c[t] * (a_c + s[t]) * (1 - s[t] - \alpha_{ws} * w^{\beta}) - y_g * s[t] - I[t] * f(s; a_s, b_s)$

The response curve is a monotonically increasing function bounded above by 1; if $b > 1$, it is sigmoidal. The parameter a controls the location of the point where f is half its maximum value, and b controls the steepness of the increasing portion. The larger the value of b, the more rapid is the switching.

$f(k;a,b) = k^b / (a^b + k^b)$

The crown biomass c grows at rater_{cs} and dies at a rate 1. The grass growth is dependent on the presence of the crown.

$c[t+1] = c[t] + r_c * s[t] - c[t]$

The fire consumption index captures the consequences of fire. A fire will break out when the grass biomass s grows a little beyond a_I. The termδ_I denotes the rate at which the fire begins to die out. The parameterr_I represents the rate of increase of the fire consumption index once sufficient fuel is present.

$I[t+1] = I[t] + I[t] * r_I * (f(s[t]; a_I, b_I) - \delta_I)$

Woody shrubs are simply defined as a logistic growth function, wherer_w represents the intrinsic growth rate of shrubs. Furthermore, fire can consume woody shrubs as denoted by the last term of the equation:

$w[t+1] = w[t] + r_w * w[t] * (1 - w[t]) - y_{Iw} * w[t] * f(I[t]; a_w, b_w)$

We implemented the model in Netlogo [download here]. We consider modeling one paddock split up into 100 cells of about 20 ha each. Each cell contains the difference equations described above using a time step of 1/50 year, about a week. Using such a small time step enables us to approximate the differential equation model results (Figure 2). In Figure 3, you see the phase diagram for a uniform grazing pressure$y_g = 0.25$. The model follows a stable

cycle of about 9-10 years. The grass biomass and the shrub biomass grow until enough fuel is available to start a natural fire. The fire consumes the grass biomass almost completely and reduces the shrub biomass significantly.

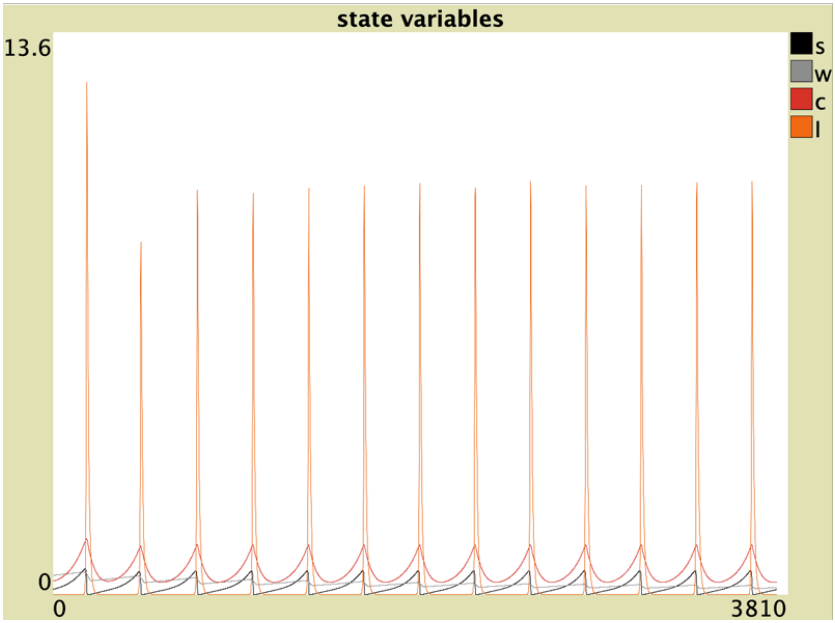

Figure 2. Simulation of the rangeland model. Each step is 0.02 years (about one week).

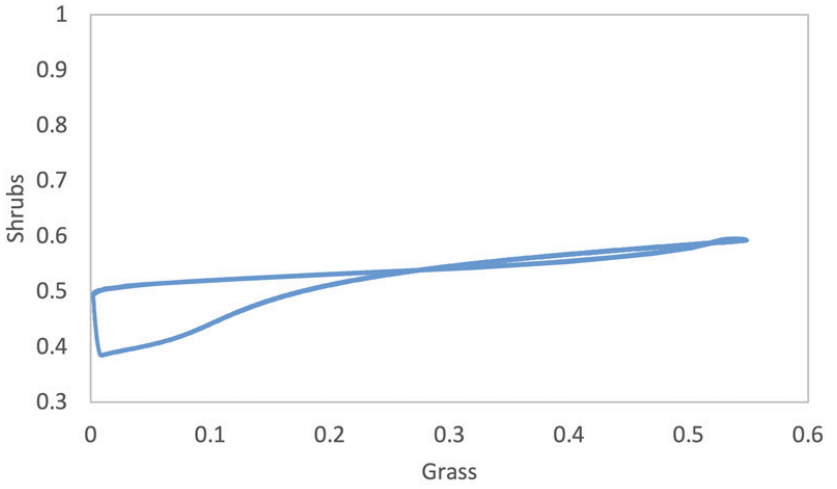

Figure 3. The trajectory that grass and shrubs follow due to periodic fires. This figure is based on the assumption of uniform grazing pressure.

We will introduce agents, representing sheep, who will harvest on individual patches and may move each time step to another patch. We will allocate 300 agents, a typical density for a property. We will explore the model with a number of different possible implementations and explore which grazing pressure the grass will be overharvested. We do this by running the model 60,000 time steps and calculate the mean for the last 10,000 time steps. Due to the model's stochasticity, we will now run the model 100 times to calculate the metric for each grazing pressure.

1. The first agent-based version assumes that all the agents move to a random neighboring patch in the next step: Move one-of neighbors

2. Instead of moving randomly, we now assume that sheep moves to the best of the nine patches centered on its current position every time step. The resulting grazing becomes similar to uniform grazing, especially since all cells start with the same conditions and have the same parameter values.

Next, we include herding behavior of sheep, assuming that sheep

have a greater tendency to occur in larger flocks. This is implemented by defining an indicator of the attractiveness of the neighboring cells. This indicator considers the amount of grass biomass and the relative distance to the water points and the number of sheep in the neighboring cells:

*att[t] = exp[-λ * s[t]) * s[t] * d + (1 – exp(-λ * s[t])) * p[t]*

and where *att* is the relative attractiveness, d $\in$ [0,1], the relative location of water point (d = 1 is the water point, d < 1 means not at the water point. The lower d the further away from the water point). p is the percentage of neighboring cells where other sheep are located. Including herding (with λ = 0.04) leads to a higher concentration of grazing pressure and, therefore, increasing the paddock's degradation for a given grazing pressure.

3. In this version, we include herding without a water point.

4. In this version, we include herding with a water point in the center of the paddock.

When we have the model according to the mean-field equation model, assuming a uniform spread of the grazing pressure, the model experience a bifurcation around grazing pressure 0.32. This means (see Figure 4) that if the grazing pressure increases to 0.32, the shrubs will dominate the landscape, and grass will disappear from the landscape due to overgrazing and competition with the shrubs.

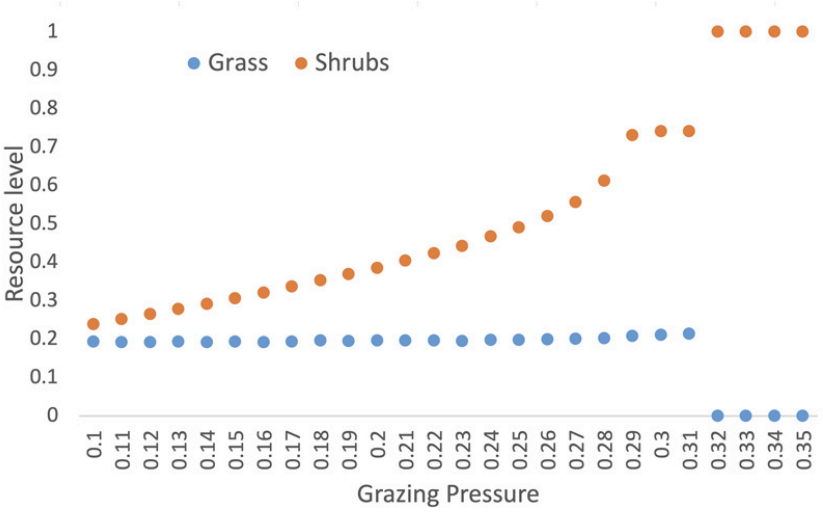

Figure 4. Long term levels of grass shoots and shrubs for different levels of grazing pressure (mean-field equation).

Figure 5 below shows that when we have sheep moving each timestep randomly to of the neighboring patches, the collapse of the grass biomass happens with a grazing level equal to 0.26. This means that the system is more vulnerable when grazing pressure is not uniformly distributed. This is not surprising. With a non-uniform distribution, some patches have a higher grazing pressure and may experience a local collapse of the grass biomass that they cannot quickly recover from if the grazing pressure is reduced.

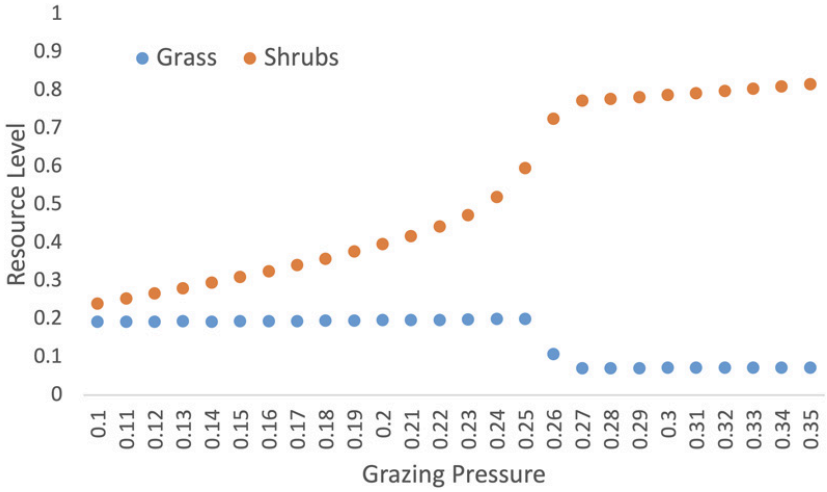

Figure 5. Long term levels of grass shoots and shrubs for different levels of grazing pressure (going to random locations).

Figure 6 below shows that when we have sheep moving to the best spots on the landscape, the collapse of the grass biomass happens with a grazing level equal to 0.22. This means that the system is more vulnerable when grazing pressure is randomly moving around. This can be explained. With agents all flocking to the best patch, grazing pressure is concentrated in a few patches, and those selected patches may local collapse of the grass biomass that they cannot quickly recover from if the grazing pressure is reduced.

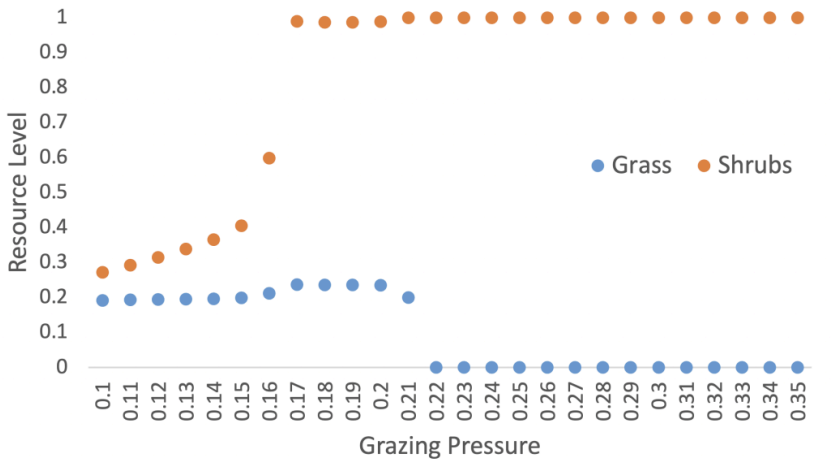

Figure 6. Long term levels of grass shoots and shrubs for different levels of grazing pressure (going to the best locations).

When we explore additional complexities such as the herding of sheep and water points in the landscape, this will not change the dynamics substantially compared to the case where sheep move to the best patches. The model shows that an agent-based model can evaluate how spatial grazing strategies can affect the resilience of the rangeland system.

Management of a Landscape of Farmers

Using the basic model of rangelands described in the previous section, we will now define the management problem as a problem of a pastoralist adjusting the livestock size and fire suppression to maximize its profit. The model is implemented in Netlogo and can be found here. We will use the mean-field equation for a property of a pastoralist. As we have demonstrated in the previous section, we know that this is a simplification that could overestimate the resilience of the system. However, instead of fixed grazing pressure on the property, we allow the pastoralist to adjust the grazing pressure if ecological conditions change.

Figure 7 provides an overview of the interactions of the different

components of the model. Grass and shrubs are competing for water and nutrients. The pastoralist adjusts the grazing pressure, which impacts the competition between grass and shrubs since livestock eats grass and not the shrubs. The pastoralist also can influence fire by the level of fire suppression. Fire consumes both grass and shrubs, but suppression could avoid fire consuming the resources too frequently. The decisions of the pastoralist on the level of grazing and the level of fire suppression are motivated by the aim of profits.

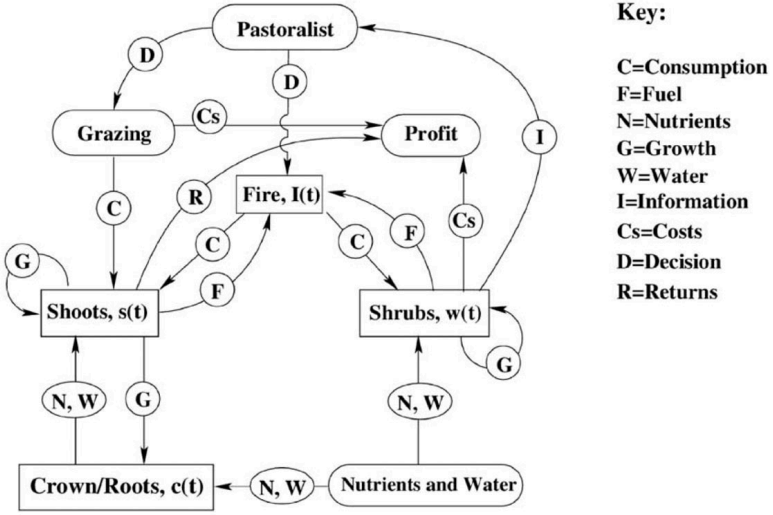

Figure 7. Relationship between the state variables and decision variables captured by the model of Janssen et al. (2004). Square boxes contain state variables with their respective symbols. The letters in the circles indicate the actions or flows indicated in the key, e.g. fire consumes shrubs and grass shoots which both provide fuel for fire. Decisions affect grazing pressure and fire as shown by the arrows flowing from the pastoralist. These decisions are made based on the information flow from the state of the shrubs

The livestock, which affects the consumption of shoots, is controlled using a rule of thumb for the pastoralist. We assume that there is a maximum number of sheep per unit area the pastoralist

will allow, z_{max}. The pastoralist uses a control variable, q_z, which defines the shrub density beyond which the pastoralist will reduce grazing pressure. If q_z is high, pastoralists reduce grazing pressure at a higher level of shrub density. The parameter b_{zmax} controls the sharpness of the response of the pastoralist as the threshold q_z is approached.

$z[t] = z_{max} * (1 - f(w[t], q_z, b_{zmax}))$

We now define the amount of shoot biomass removed by grazing as the multiplication of z with $s/(s + 0.1)$, a response function in line with Holling's disc equation type II.

With no manager or sheep, fires will die out naturally when the fuel load is consumed. With increasing numbers of sheep on the property, the pastoralist must suppress fire to provide feed for the stock and maintain wool production. If z increases, the pastoralist would increase from its minimum value of λ up to a maximum of 1. The control variable q_δ defines a threshold stocking rate at which the pastoralist begins to suppresses fire, and b_δ defines the sharpness of the pastoralist's response as this threshold is approached. As such we define δ_l as

$\delta_l[t] = \lambda - (1 - \lambda) * f(z[t], q_\delta, b_\delta)$

Building on Janssen et al. (2004), we define the profit of a pastoralist as a function of wool production, the cost of mustering sheep, adjustment costs associated with changes in grazing pressure, and lost revenue when grass during a fire. The first two elements are driven indirectly by grass and shrub dynamics in the system. The second two are direct consequences of management action and are thus related to the choice of control parameters.

Wool production is assumed to be a function of the number of sheep and their nutritional status measured by grass offtake. When grass biomass is low, offtake will be low, and the animals will be malnourished. This will cause wool production to go down. In the semi-arid region of Australia, supplementary feeding is not economically feasible. When shoot biomass is high, offtake will be high, sheep will be well-nourished, and sheep numbers will be

the primary determinant of wool production, and wool production varies with z: Costs associated with shrubs are related to mustering—locating and gathering animals on several occasions per season. The more shrubs, the more difficult and costly it is to muster the animals. The cost per animal associated with shrubs does not increase significantly until shrub density is quite high. A first-order approximation to capture this fact is a quadratic representation. This yields a cost per unit of grazing pressure of $C * w^2$.

The cost of adjusting the sheep density (moving sheep on or off the property), C_a; is directly related to the derivative of the sheep density. Assuming a constant movement cost per sheep (agistment costs), we assume that changes in the stock level are due entirely to management actions. This, of course, is not entirely accurate. However, given that managers decide when to mate their ewes and rams, when to buy, sell, and move their stock, they assume that these adjustments are stronger determinants of the stock dynamics than natural population dynamics.

Now we can define the profit function π[t] in time step t as

$\pi[t] = z * (f(s[t]; 0.1, 1) - C_w * w[t]^2) - C_a * |\Delta z|$

What is the best management policy for the rangeland system? We can formulate this as an optimization problem where a pastoralist maximizes the accumulation of discounted profits given z_{max},q_z, and $q\delta$ being the control variables.

We can solve this problem in Netlogo using the genetic algorithm from the BehaviorSearch tool of Netlogo. When we use an annual discount rate of 2%, we that the optimal values of the control parameters are 0.324 for z_{max}, 0.582 forq_z, and 0.347 for q_δ. This leads to a 3-year cycle of the system in which the stocking is adjusted between 0.33 and 0.36, and the grass biomass is kept at a high level due to fire suppression. In Figure 8, we see that the stocking rate declines when the biomass of shrubs is increasing. But because the control strategy keeps the shrub level contained, the stocking rates vary not that much either.

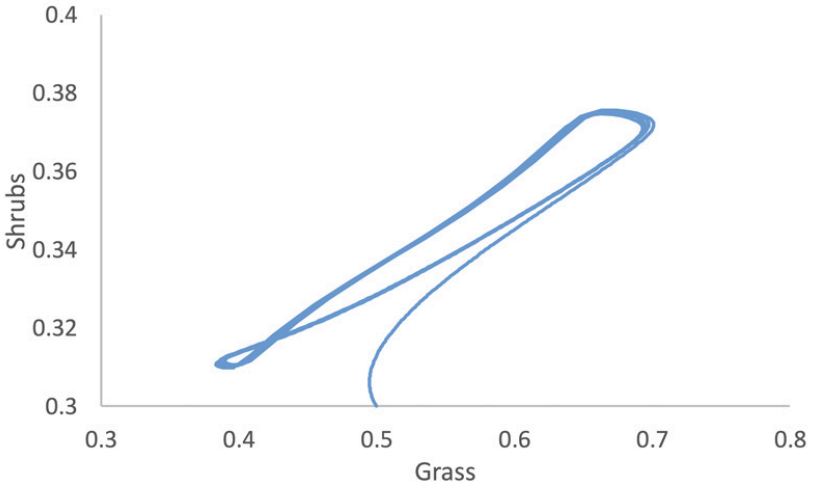

Figure 8. The optimal strategy depicted in a phase plane of grass and shrub levels. The system will go through periods of recovery (increasing amount of grass and shrubs) and fire but stays within a system of co-existence of grass and shrubs.

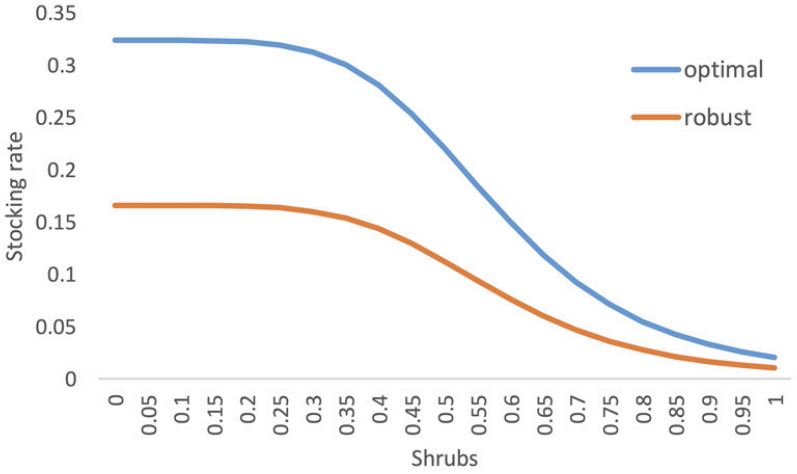

Figure 9. The control strategy for stocking levels as a function of shrub biomass. The optimal strategy is the outcome of the optimization without stochastic events. The robust solution is the outcome of the optimization with stochastic rainfall events.

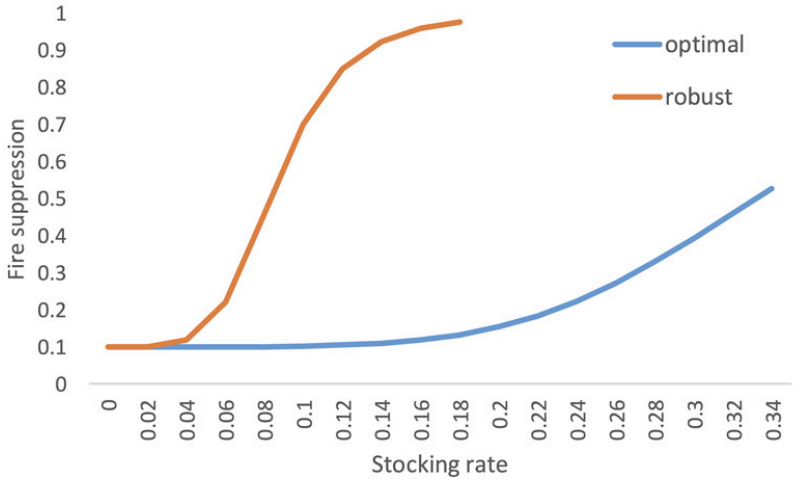

Figure 10. The control strategy for fire suppression levels as a function of the stocking rate. The optimal strategy is the outcome of the optimization without stochastic events. The robust solution is the outcome of the optimization with stochastic rainfall events.

We now include rainfall variability in the model. Variability in rainfall patterns increases the vulnerability of the system to grazing pressure. In wet years, the shrub growth is faster. This, combined with heavy grazing pressure, reduces the accumulation of shoot-biomass, which will increase the chance that the system will fall into the stability domain dominated by shrubs. In this case, more caution is needed to manage shrubs when rainfall is variable. Rainfall is defined which will affect growth rates of crowns, shoots, and shrubs, and follows a lognormal distribution with mean 1 and variance 0.28:

$r_f = exp(n(-0.125, 0.5))$

When we use a genetic algorithm to find an optimal solution, we calculate the value of the objective function as the mean of 100 runs with different rainfall patterns. The resulting optimal solution can, therefore, be considered as a robust solution leading to, on average, a high mean value of accumulated discounted profits for

diverse rainfall patterns. The maximum stocking level is halved (z_{max} =0.166 and q_z =0.58), and fire suppression is much higher at lower stocking levels (q_δ =0.087).

We now switch gears and create a landscape of N pastoralists who each have their own property. We assume that they all have the same initial conditions of the ecosystem, and all have the same equations, as above, of the ecological dynamics. However, the pastoralists differ in their values of z_{max},$q_{z,}$ and q_δ. Every year, pastoralists evaluate their profit level with the profit levels of their neighbors. If a neighboring pastoralist has better performance, the management style (values of z_{max}, $q_{z,}$ and q_δ) are copied, with a random value added (drawn from n(0, 0.05).

When we run the model 100 times for 1000 years, the mean median for the management style is 0.15 (0.07) for z_{max}, 0.42 (0.08) for$q_{z,}$ and 0.12 (0.10) for q_δ, where the values between brackets are standard deviations for the sample of 100. The initial management styles are similar to robust strategies with our optimization analysis. We realize that a 1000 year period is unrealistic, but we use this extreme length to look at long term effects of an evolved process.

Compared to the optimal robust strategy, agents have a similar maximum stocking rate but are reducing its stocking level with a lower level of shrub and have less fire suppression.

The next step is to evaluate the impact of policy interventions on the management strategies that evolve. Do policies lead to different management styles of pastoralists? We consider two policies. The first policy is a so-called conservation policy. The pastoralist is requested to destock their property if grass cover is low, namely is $s < s_{min}$. Whens_{min} is higher, this means that properties are stocked more frequently. We see that for higher values ofs_{min} the maximum stocking rate is higher and that the reduction of destocking rate is with higher levels of the shrub. Furthermore, we observe less fire suppression. As such, for higher values of s_{min}, there is more intense grazing.

Table 1. The emerged strategies for the conservation policy for different threshold values for s_{min}.

s_{min}	z_{max}	q_z	q_δ
0.05	0.20 (0.08)	0.55 (0.09)	0.08 (0.05)
0.10	0.24 (0.08)	0.67 (0.07)	0.10 (0.03)
0.15	0.26 (0.08)	0.72 (0.07)	0.13 (0.02)
0.20	0.28 (0.07)	0.74 (0.06)	0.16 (0.03)
0.25	0.33 (0.08)	0.75 (0.10)	0.19 (0.03)
0.30	0.36 (0.09)	0.79 (0.09)	0.21 (0.05)
0.35	0.41 (0.09)	0.77 (0.09)	0.24 (0.05)
0.40	0.42 (0.08)	0.73 (0.14)	0.26 (0.06)
0.45	0.39 (0.13)	0.73 (0.16)	0.22 (0.07)
0.50	0.41 (0.10)	0.74 (0.08)	0.18 (0.05)

The second policy is a drought relief, where pastoralists will be asked to destock their land if there is a drought $r_f < r_{fmin}$, and the pastoralists will get a compensation equal to subs * Δz. When drought relief is implemented more often (higher value of r_{fmin}), the maximum stocking rates are higher, and there is less fire suppression. If governments give more subsidy, there is also more stocking. Hence drought relief policies can lead to very high stocking rates during times when there is no drought emergency.

Table 2. The emerged values for z_{max} for drought relief policies defined by subsidy levels and r_{fmin}.

Subsidy\r$_{fmin}$	0.1	0.3	0.5	0.7	0.9
0.1	0.14 (0.08)	0.15 (0.08)	0.17 (0.10)	0.28 (0.14)	0.81 (0.13)
0.3	0.15 (0.09)	0.19 (0.12)	0.70 (0.17)	0.95 (0.04)	0.98 (0.01)
0.5	0.16 (0.10)	0.31 (0.23)	0.96 (0.03)	0.97 (0.02)	0.98 (0.01)
0.7	0.15 (0.12)	0.89 (0.09)	0.97 (0.02)	0.98 (0.01)	0.98 (0.01)
0.9	0.16 (0.13)	0.96 (0.03)	0.97 (0.01)	0.98 (0.01)	0.98 (0.01)

Table 3. The emerged values for q_z for drought relief policies defined by subsidy levels and r_{fmin}.

Subsidy\r$_{fmin}$	0.1	0.3	0.5	0.7	0.9
0.1	0.42 (0.07)	0.44 (0.07)	0.50 (0.08)	0.68 (0.17)	0.97 (0.02)
0.3	0.46 (0.07)	0.58 (0.13)	0.96 (0.03)	0.98 (0.01)	0.99 (0.01)
0.5	0.47 (0.09)	0.62 (0.29)	0.98 (0.01)	0.98 (0.01)	0.99 (0.01)
0.7	0.47 (0.14)	0.97 (0.02)	0.98 (0.01)	0.99 (0.01)	0.99 (0.01)
0.9	0.47 (0.16)	0.98 (0.01)	0.99 (0.01)	0.99 (0.01)	0.99 (0.01)

Table 4. The emerged values for q_δ for drought relief policies defined by subsidy levels and r_{fmin}.

Subsidy\r$_{fmin}$	0.1	0.3	0.5	0.7	0.9
0.1	0.11 (0.08)	0.14 (0.08)	0.15 (0.11)	0.16 (0.12)	0.18 (0.14)
0.3	0.12 (0.10)	0.14 (0.10)	0.21 (0.16)	0.22 (0.16)	0.19 (0.15)
0.5	0.12 (0.10)	0.20 (0.17)	0.20 (0.15)	0.20 (0.13)	0.20 (0.14)
0.7	0.14 (0.12)	0.23 (0.16)	0.20 (0.15)	0.21 (0.16)	0.17 (0.14)
0.9	0.15 (0.11)	0.25 (0.17)	0.23 (0.17)	0.19 (0.14)	0.17 (0.13)

Possible Extensions

There are various ways the model can be extended. Besides applying the model to an empirical landscape, there are interesting extensions with the stylized version of the model. One assumption we made in the model was that rainfall was uniform on the

landscape, but that is obviously not the case. Rainfall is spatially correlated, but not uniform. This has important implications. Parts of the landscape may experience a drought while others have sufficient rainfall. In McAllister et al. (2006), we explored the consequences of agistment arrangements between pastoralists. A pastoralist who experiences a drought may find a pastoralist with healthy grass biomass, move its livestock to that property, and hopefully get it back a few months later when conditions on the property of the first pastoralist have improved. We could look at this problem as a trust game where the pastoralists have to trust each other that no diseases are carried over by moving livestock and that the moved livestock will survive and returned back.

As such, this could lead to social networks between pastoralists that are driven partly by spatial patterns of rainfall density. It is not uncommon that livestock is moved hundreds of miles in Australia to park livestock from areas with a severe drought.

Another element that requires attention is the kangaroo population. Due to the increase of water points in the landscape for the livestock, the kangaroo population has a higher carrying capacity. Furthermore, kangaroos are not limited to property lines since they jump over fences. In fact, experiments have been done on how high fences need to be, around a costly 3 meters, to reduce the movement of kangaroos between properties. As a consequence, a pastoralist who is doing well to maintain her property may welcome additional grazers, and may not experience all the benefits it could have.

In future versions of this book, new versions of the model with those extensions might be included, but I also challenge the readers to create new variations of the model.

Summary

In this chapter, we presented a series of models on rangeland management that combined difference equations with small time steps mimicking the differential equations of the nonlinear ecological dynamics. With this, we could have agents adjusting the

grazing pressure on the patches (sheep movement and grazing pressure of the property). We used BehaviorSearch to find optimal strategies for this non-linear stochastic system. We showed that we could discover behavioral strategies for the pastoralists, which keep their system in a productive state.

References

Anderies, John M., Marco A. Janssen, and Brian H. Walker (2002) Grazing management, resilience, and the dynamics of a fire driven rangeland system, *Ecosystems* 5: 23-44.

Holling, Crawford S. (1973) Resilience and stability of ecological systems, *Annual Review of Ecology and Systematics* 4: 1–23

Janssen, Marco A., Brian Walker, Jenny Langridge, and Nick Abel (2000) An adaptive agent model for analysing co-evolution of management and policies in a complex rangeland system, *Ecological Modelling* 131(2/3): 249-268.

Janssen, Marco A., John M. Anderies, Mark Stafford Smith, and Brian H. Walker (2002) Implications of spatial heterogeneity of grazing pressure on the resilience of rangelands, in Janssen, Marco A. (ed.) Complexity and Ecosystem Management: The Theory and Practice of Multi-agent Systems, Edward Elgar Publishers, Cheltenham UK/ Northampton, MA, USA. Pp 103-124.

Janssen, Marco A., John M. Anderies and Brian H. Walker (2004), Robust strategies for managing rangelands with multiple stable attractors, *Journal of Environmental Economics and Management* 47:140-162.

McAllister Ryan R.J., Ian J. Gordon, Marco A. Janssen, and Nick Abel (2006), Pastoralists' Responses to Variation of Rangeland Resources in time and space, *Ecological Applications* 16(2): 572-583.

Westoby, Mark, Brian H. Walker, and Imanuel Noy-Meir (1989) Opportunistic management for rangelands not at equilibrium. *Journal of Rangeland Management* 42, 266–274

PART V

DIFFUSION AND SOCIAL INFLUENCE

In this part of the book, we will introduce networks, and how genetic or social information is spread through networks.

CHAPTER 17

Networks

Key Concepts

In this chapter, you will

- be introduced to the concept of networks,
- be presented different types of networks such as random network, small-world network and scale-free network,
- be exposed to metrics to compare networks such as degree, path length, and clustering coefficient.

In previous chapters, the interactions were often defined by random interaction, the spatial proximity of moving agents, or spatial proximity of stationary agents. In this chapter, we focus on agents interacting within social networks. An excellent example of a social network is Facebook, where people interact with others because of their social connections, not because of spatial proximity. Those interactions are also not random since when we analyze the structure of those social interactions, we observe phenomena not observed in random networks, as we will discuss below.

The study of networks is the study of the structure of

relationships between nodes (often agents in our models), which affect the outcome of processes such as the spread of rumors or diseases. If a well connected Facebook member puts a rumor on her page, this information will spread much faster among the Facebook community compared to the case in which the same rumor is placed on the page of a person who has only a small number of not very active friends. But what do we mean with "well connected." In this chapter, we will present a number of network metrics that can be calculated to define objective measurements of network structure that can help understand how structure impacts the processes on the network. We will discuss various applications in the coming chapters. In this chapter, we will discuss the basics of networks.

Basics of Networks

A network is a collection of nodes that are connected by links. A node can represent a person, a family, a nation, a computer, a species, etc. Links define a relationship between nodes such as friendship, kinship, a highway, who-is-eating-whom, etc. In a NetLogo model, we will have turtles defined as the nodes. The links are defined as a spatial category link next to patch and turtle.

Links can be directed. A link representing agent A is the parent of agent B is a direct link (Figure 1). But a link that represents a friendship between agents A and B can be an undirected link. That is, if both agents see each other as friends.

Figure 1. Directed and undirected links between agents A and B.

Each agent can have a number of links. The number of links an agent has is called degree. If an agent has five undirected

connections, we would say that the agent's degree is five (Figure 2). In some networks, the degree is the same for each agent, while in others, the degree is different for each agent. The distribution of degrees in a population of agents us an important characteristic of a network.

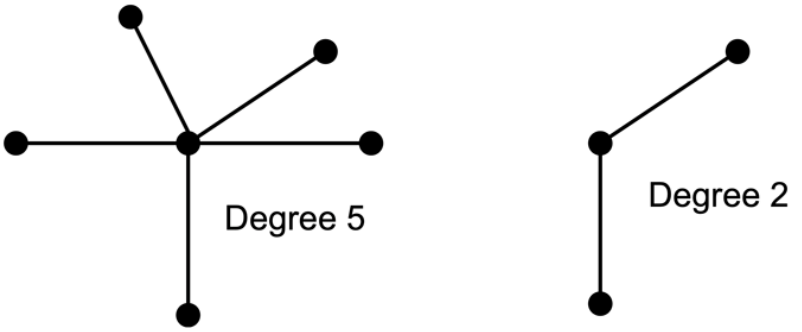

Figure 2. Central node gas degree 5 (left) and 2 (right).

The paths between nodes defined another characteristic of networks. In the figure below, you see a path between the two red nodes. There are various paths possible. There are even different shortest paths of four links between the red nodes. We can calculate the average shortest path between every two nodes of a network. This leads to the average shortest path length of a network. Network structure affects the length of the shortest path, and therefore it becomes an important indicator to compare networks. The shorter the average path length, the faster we will expect information will diffuse within a network.

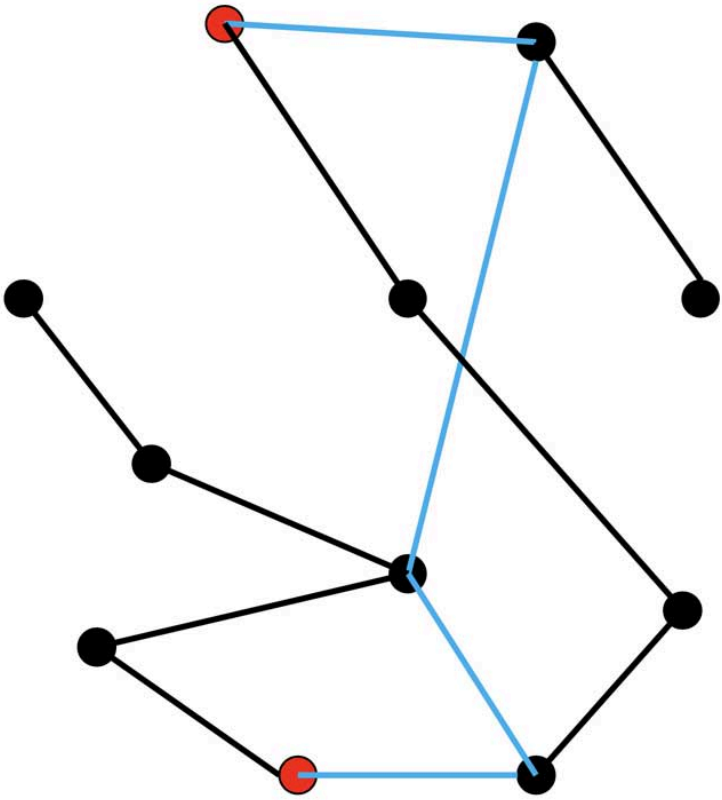

Figure 3. Shortest path – 4 links – between two red nodes.

Finally, we look at the clustering coefficient of a network. This coefficient indicates how frequently links of a node are linked with each other. In social networks, we can think of friends of person A also being friends of each other. A higher amount of friends of friends leads to more clustering. In the figure below, you see that the red node has three (yellow) friends, but none of these friends are connected. Thus the clustering coefficient of the red node is zero. To calculate the clustering coefficient for a network, we take the average of the clustering coefficients for all nodes.

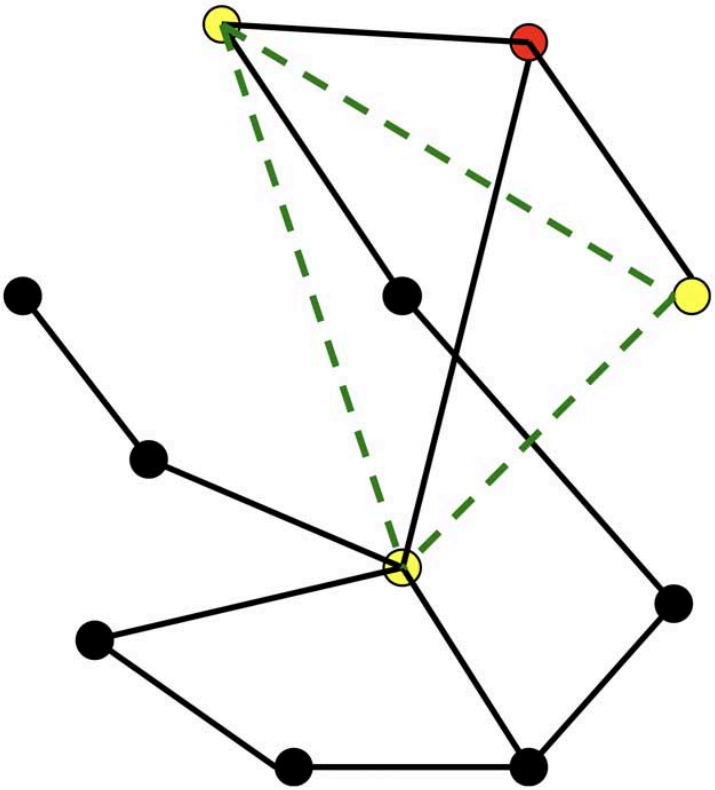

Figure 4. Dotted green lines are missing connections between 3 yellow friends of the red node.

A network is visualized as a graph. Often these graphs have no spatial representation. It has when it represents highways or other physical networks. Otherwise, these graphs are just a graphical representation of the relationships. We will now discuss a number of different types of networks and see how they differ in the network metrics degree, path length, and clustering.

Random Networks

Let's start by looking at a simple network, the random network. In

this network, there is a population of n nodes. One by one, we add links between nodes. We do this by randomly picking two nodes and linking them if there is no link already.

In the NetLogo code (Giant Component in the Networks folder of the NetLogo library), you see that adding a random link is done as follows. Pick two turtles and check whether there is already a link between them. To do this, we make use of link-neighbor? X, which is a primitive that is true if there is a link between the turtle and X. If there is no link already, we can create a new link between the two turtles using create-link-with X. The procedure add-edge is a recursive loop that keeps looking for nodes that are not yet connected before adding a new link.

```
to add-edge
  let node1 one-of turtles
  let node2 one-of turtles
  ask node1 [
    ifelse link-neighbor? node2 or node1 = node2
    [ add-edge ]
    [ create-link-with node2 ]
  ]
end
```

In Figure 5, you see how the network is visualized during the construction of links. The red links and turtles are the largest connected component of the network.

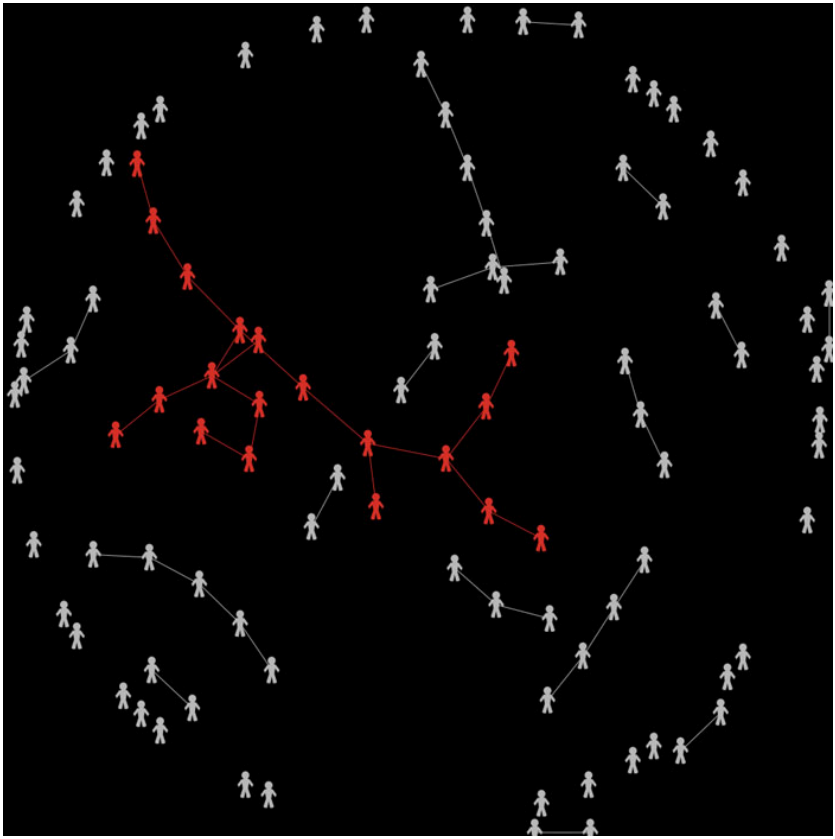

Figure 5. Creating a random network.

We can visualize the relative size of the biggest component over time (Figure 6). Initially, the size is very small, just a few turtles. Then there are jumps, as small components get connected. It takes a while until all turtles are connected.

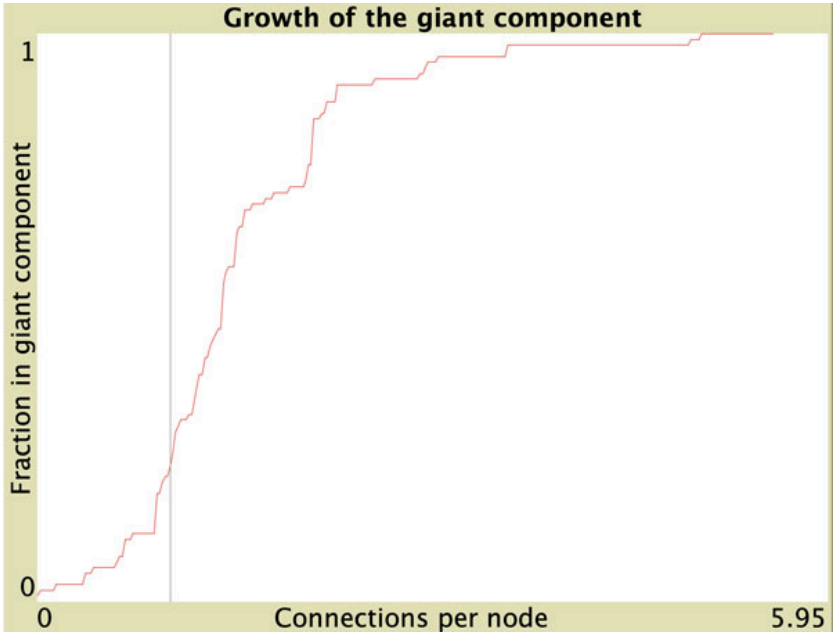

Figure 6. The fraction of the biggest component of connected turtles as part of the whole network. The vertical line is the moment in time where there are as many links as turtles.

Not all turtles have the same number of links (Figure 7), but they will each have a minimum of 1 since the model ran until all turtles were connected. In the example simulation, we see that the number of links varies from 1 to 12. The median degree is 4. The clustering coefficient is 0.054 and the average path length is 2.96 links.

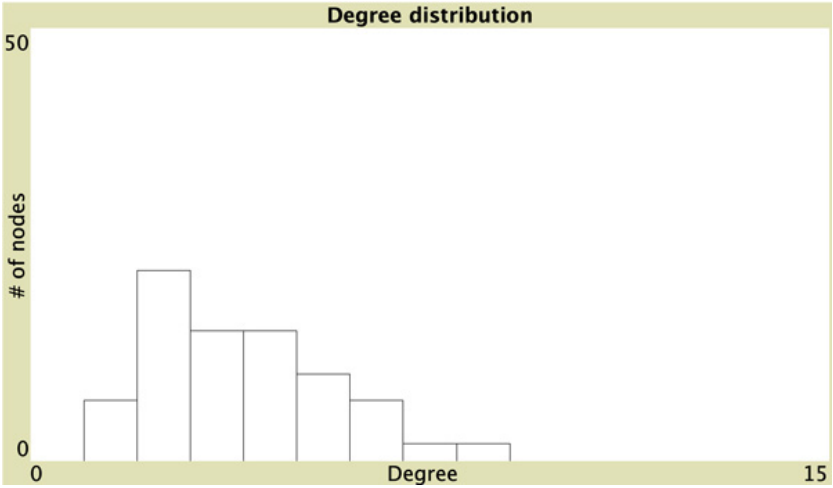

Figure 7. The distribution of degrees in the example run.

Small-World Networks

One may have experienced an encounter with a stranger linked within just a few connections with you. In this way, one might be just a few handshakes away from the President of the United States. Let's define "handshake" here are literally shaking someone's hand as you meet in person (or any alternative greeting ritual during a pandemic). Let's exclude the handshakes Presidents give to the crowd when they have rallied. So, how many handshakes are you away from the President? You may be surprised at how small this number is for most people. Let's do this with a person picked at random from the billions of adults living on this planet and try to define how many handshakes they are away. Any guess? Here we would expect that this number is not much more than a handful of handshakes.

This suggests that the average path length of social networks around the world is small. Psychologist Stanley Milgram did an experiment in the 1960s where he asked a number of people in the midwestern United States to send a letter to somebody in Boston. However, they did not receive the person's address, but

just a number of hints, like an occupation. To reach their target, people had to send the letter to somebody they thought would have more success in finding the target. Milgram was interested in whether the letters would arrive at their target address and how many connections it would take. Not all letters reached their target, but of those that reached the target, the average number of links was six. This led to the saying that all people are, on average, just six handshakes away. The experiment was replicated 35 years later with emails, and again the average number of links between the source and the target was found to be six (Dodds et al., 2003).

Is the random network a good model for social networks? No, since social networks have another distinctive attribute. Many friends of a person are also friends of each other. This is called clustering within networks. To calculate the clustering coefficient of a node, we will look at all the friends of this node, and count how many possible friendships exist between nodes. The total actual friendships among friends divided by the total possible friendships. This is the clustering coefficient, which was 0.054 in the random network example. In random networks, this coefficient is not very high since many connections are random, and friends are not friends of each other.

We will now look at the small world network that captures both the high level of clustering and a short average path length. The model was developed by Duncan Watts and Steven Strogatz. They started with a regular network of nodes that are arranged in a circle (Figure 8). Each node is connected to two nodes to the left and two nodes to the right. This means that the clustering coefficient is 0.50 since 50% of the links between the friends of a node are connected with each other. For a network of 100 nodes, the average path length is about 13 links.

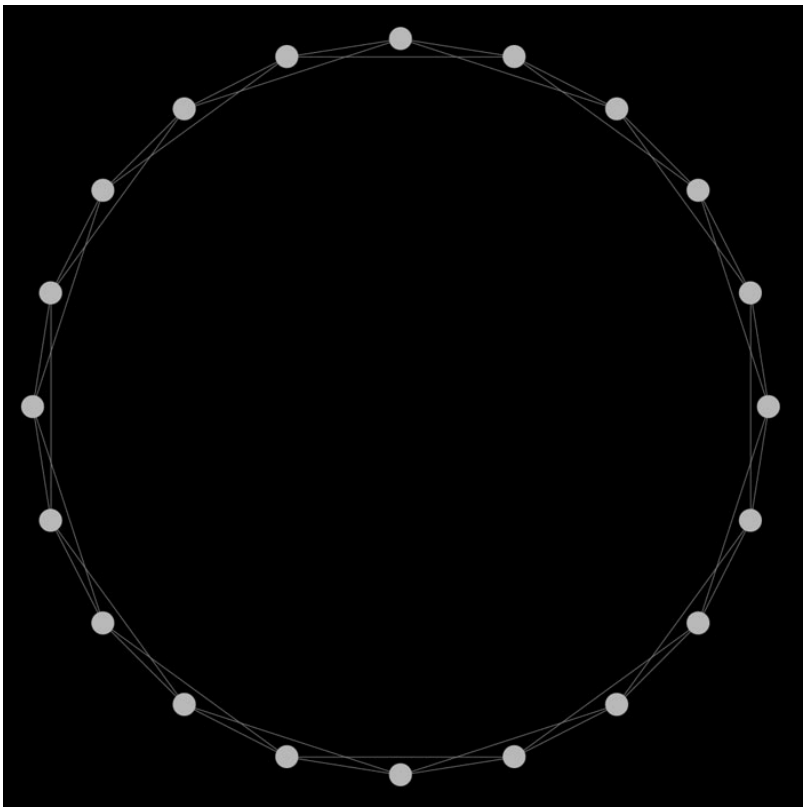

Figure 8. A regular network of 20 nodes where each node is connected with two nodes on the left and two nodes on the right.

When we relink a few nodes, the average path length drops significantly. In the figure below, the relink of four nodes leads to a drop in the average path length from 13 to 9 links. The relink a node, we chose a random node A and then chose randomly from this node one of the links with another node B. The node B is then replaced with a randomly chosen node from the whole population of nodes, except node A and existing nodes of node A.

We will now look at the NetLogo implementation of Small Worlds in the Networks folder of the NetLogo library. A new primitive we

use is end1, which indicates the first turtle of a link. In a directed link, this will be the source, and for undirected links, it will be the turtle with the lower who number. We first check whether new links can still be added to the turtle of interest. If this is the case, a turtle is drawn who is not yet connected to the turtle of interest. We then create a link between both turtles. In Netlogo this looks like

```
let node1 end1
if [ count link-neighbors ] of end1 < (count turtles - 1)
[
    let node2 one-of turtles with [
       (self != node1) and (not link-neighbor? node1)
    ]
    ask node1 [ create-link-with node2]
]
```

When we start relinking, we will make cross-cuts from one part of the network to another part (Figure 9). This will lead to shorter paths between the ends of the network.

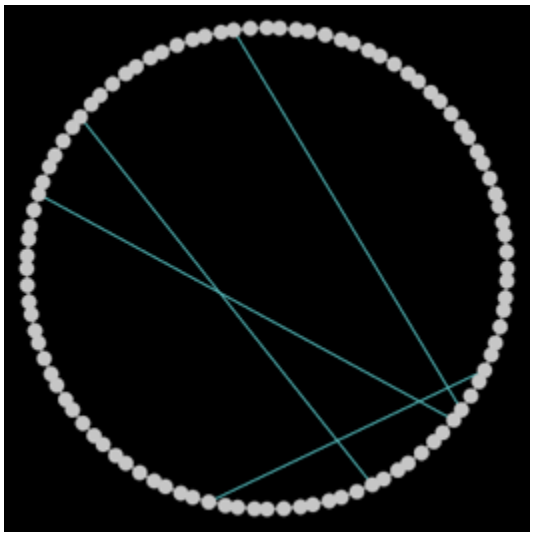

Figure 9. Relinking eight nodes by four links of a regular network.

If we relink many nodes, the average path length will initially drop quickly. The average cluster coefficient stays initially high. As shown in Figure 10, the shortcuts reduce the path length between nodes from both sides of the network but do not affect the links among friends. As we see in Figure 10, just a few relinkings lead to networks with high clustering and a low average path length. This is the characteristic of small-world networks. If we relink all the nodes, we will end up with a random network with low average path length and a low amount of clustering.

What we call a small-world network is a network with just a small fraction of short cuts. This leads to a high clustering coefficient and a short path-length. This is also the kind of network that we find in social networks. Most people are clustered, and friends of friends are friends. But if just a small percentage of the friendships are with very different people, the average path length will be small.

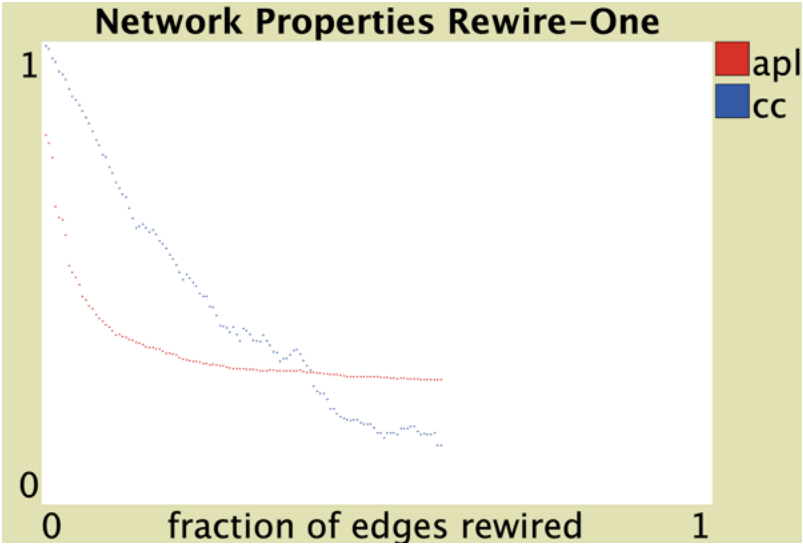

Figure 10. Relinking eight nodes by four links of a regular network.

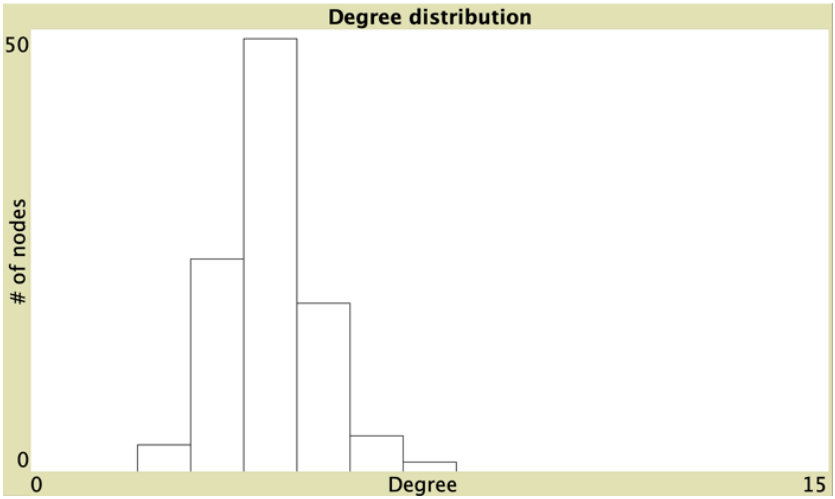

Figure 11. The average path length (apl) and clustering coefficient (cc) for an increasing number of nodes relinked in a network of 100 nodes.

Scale-Free Network

The final cartoon network we will discuss is the scale-free network. The term cartoon network is used since actual networks do not exactly follow the idealized networks we are modeling. More detailed simulation models need to be used to capture more detailed network characteristics of specific networks.

The scale-free network captures the observation that degree distribution is not normally distributed in many real-world networks, as seen in Figures 7 and 11. For example, many websites do not link to others, but some have many links. This last category is considered to be a hub. If we plotted the degree distribution on a log-log scale, we would see a straight line. If it was a normal distribution, the frequency of observations with a high in-degree, say 100, would be close to zero. This means that in a scale-free distribution, the distribution has fat tails. There are more outliers than would have been in a normal distribution.

Barabasi and Albert (1999) published a simple model that generates networks that have a scale-free degree distribution. The

model is based on preferential attachment. Starting with two nodes, every new node will be linked to the existing network, but with a higher preference to nodes with a higher degree number. Thus nodes with a high degree have a higher probability of getting more nodes connected. This is also called the "rich become richer" phenomenon.

To model the process of drawing a connecting node based on the degree of the nodes already in the network, we will use the Preferential Attachment model in the Networks folder of the NetLogo library. First, we draw a random value between 0 and the total degree number in the network. In random order, the degree of turtles is subtracted from this random value until the algorithm gets to a degree of a turtle being lower than the adjusted random value initially generated. This is a partnering turtle that is drawn to be connected to the new turtles. The higher the level of the degree, the more likely that turtle will be drawn.

```
let total random-float sum [count link-neighbors] of turtle
let partner nobody
ask turtles
[
  let nc count link-neighbors
  if partner = nobody
  [
    ifelse nc > total
      [ set partner self ]

      [ set total total - nc ]
  ]
]
```

In Figure 12 you see a snapshot of the network generation in the process. The size of the nodes is proportional to the degree. We see a few hubs and many nodes that are only connected to one other node.

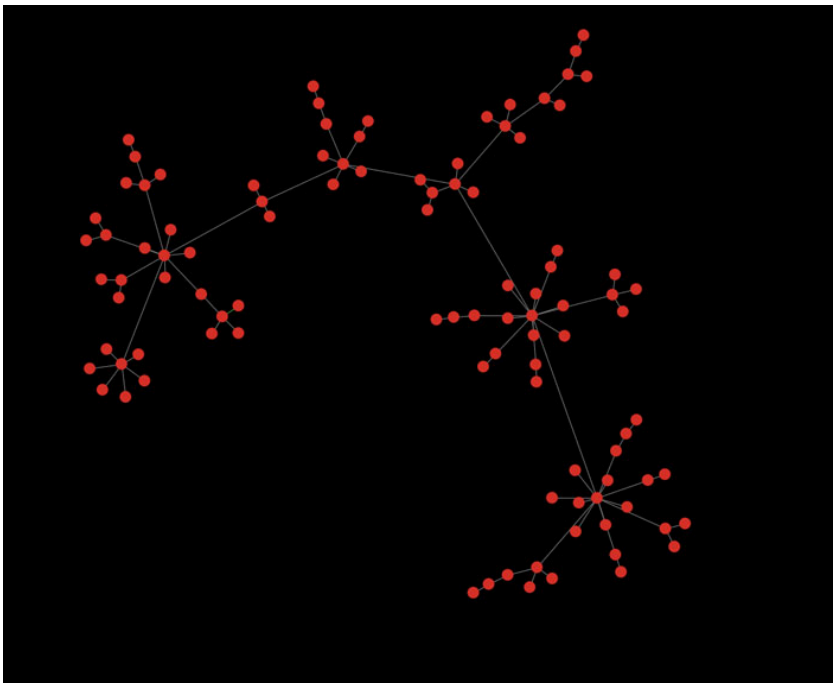

Figure 12. A typical scale-free network.

Figure 13 shows the degree distribution of the network of 100 nodes generated in the example of Figure 12. The distribution approximates a scale-free distribution, meaning a straight line on a log-log scale of the number of nodes and the degree. The network has 66 nodes with degree one, and only two nodes with a degree higher than 8.

The clustering coefficient is 0 since friends of friends are not friends. The average path length is 4.14, which is higher than the average path length when the network was randomly generated (2.96). How do we calculate those metrics? Copy the relevant procedures from the Small World model to the Preferential Attachment model. Try it out!

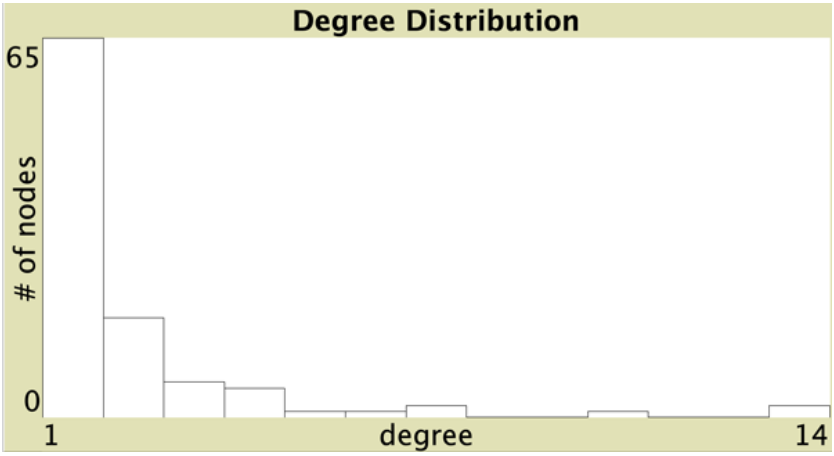

Figure 13. Degree distribution of the network in Figure 12 of 100 nodes.

Summary

Agents can be connected in networks representing social or physical relationships. In this chapter, we introduced the basics of networks and how to implement classic networks such as random, small-world, and scale-free networks in NetLogo.

References

Barabási, (2004) Linked: How Everything is Connected to Everything Else. Plume, Cambridge, MA.

Barabási, Albert-László and Réka Albert (1999) Emergence of scaling in random networks. *Science* 286: 509-512.

Dodds, Peter S., Roby Muhamad, and Duncan J. Watts (2003) An Experimental Study of Search in Global Social Networks, *Science* 301: 827-839.

Newman, Mark, Albert-László Barabási, and Duncan J. Watts [eds.] (2006) The Structure and Dynamics of Networks. Princeton, N.J.: Princeton University Press.

Strogatz, Steven H. (2001) Exploring complex networks. *Nature* 410: 268-276.

Watts, Duncan J. and Strogatz, Steven H. (1998) Collective dynamics of 'small-world' networks. *Nature* 393: 440–442.

CHAPTER 18

Epidemics

Key Concepts

In this chapter, you will

- be introduced to the SIR model to model epidemics,
- see the definition of R0, which shows whether an epidemic will happen or not,
- explore the effect of network structure on the spread of infectious diseases.

In the movie Contagion, a bird-flu-infected meal in Hong Kong causes a global pandemic killing millions of people. The lead actors try to trace down the source of the virus and calculate the variable R0, the basic reproduction number, to understand how fast the virus will spread. The COVID-19 pandemic we experienced followed a similar script. However, the reproduction rate R0 is still one of the key variables discussed to determine the severity of the virus's ability to spread.

Infectious diseases spread from human to human and can spread like wildfire. Depending on the disease, transmission can

be caused by coughing, sexual intercourse, doorknobs, toilet seats, and blood transfusion.

Infectious diseases are diffused through a population via the interactions of agents. In this chapter, we will discuss agent-based models of infectious diseases. We will show how network structure affects the spread of the virus. We can use the model to see the potential effects of interventions, like closing schools, physical distancing, wearing masks, and washing hands, to reduce the transmission of the virus.

The model we discuss in this chapter is based on a basic flu-type virus. Various other viruses contain more complex dynamics and stages of infectiousness.

SIR Model

A classic way to describe the diffusion of a disease in a population is to distinguish three components: susceptible, infected, and recovered. The so-called SIR model is typically studied using differential equations. There are three variables, S the number of susceptible, I the number of infected, and R the number of recovered. If we exclude population change and only look at the virus, we can model that the number of susceptible decreases over time. The rate of this decrease depends on the number of infected and susceptible people. The more susceptible people there are, the more people can become infected. The more infected people there are, the more they can infect susceptible people. The parameter β is the contact rate, representing the rate of infections when infectious and susceptible individuals randomly interact.

$S[t+1] = S[t] - \beta * I[t] * S[t]$

The number of infected people is increased by the number that gets infected minus the number of people who recover. People recover at a fixed rate and the recovery rate is given by γ.

$I[t+1] = I[t] + \beta * I[t] * S[t] - \gamma * I[t]$

$R[t+1] = R[t] + \gamma * I[t]$

When will there be an epidemic? That is the case when I[t] is increasing over time. The basic reproduction number, R0, can be

derived as an expression of parameters to measure when I increase over time. R0 provides the expected number of new infections from a single infectious person in the population. If R0 > 1, more people get infected with each infection, and the virus spreads, and we have an epidemic. R0 is defined as

$R0 = \beta / \gamma$

To combat the epidemic, R0 needs to become lower than 1. To do this, we need to reduce β, which means a change in behaviors to reduce the spread through interactions such as hand washing, better hygiene, and using condoms, or increase γ, which will be the improvement of medical treatments to improve recovery.

SIR Model on a Network

The difference equation-based model, as described above, assumes that many agents are randomly interacting. We now discuss an agent-based model where we distinguish susceptible, infected, and recovered agents. We will use the model "Virus on a Network" in the Networks folder of the NetLogo library as the starting point.

Given are agents connected in a network. The network is generated as follows. First, all the agents are randomly put on the landscape.

```
crt number-of-nodes
[
   setxy random-xcor random-ycor
]
```

Then we define the number of links that will be generated such that the network has a specific average degree: average-node-degree

```
let num-links (average-node-degree * number-of-nodes) / 2
```

Links are added one by one until num-links is reached. Every time step one of the turtles is selected randomly and the nearest turtle to it is selected and a link is made between them.

```
while [count links < num-links]
```

```
[
   ask one-of turtles
   [
      let choice min-one-of (other turtles with [not link-n
      if choice != nobody [create-link-with choice]
   ]
]
```

Agents who are connected may affect each other's state. Infected agents may infect susceptible agents. Each susceptible neighbor of an infected agent has the probability virus-spread-chance of getting infected. Each infected agent has a probability recovery-chance of recovering. If an agent recovers, it can become immune with a probability gain-resistance-chance or become susceptible again. This is implemented in NetLogo in the following way.

```
ask turtles with [infected?]
[
   ask link-neighbors with [not resistant?]
   [
      if random-float 100 < virus-spread-chance
      [
         set infected? true
         set resistant? false
         set color red
      ]
   ]
]

ask turtles with [infected?]
[
   if random 100 < recovery-chance
   [
      ifelse random 100 < gain-resistance-chance
      [
         set infected? false
         set resistant? true
         set color gray
      ][
         set infected? false
         set resistant? false
```

```
            set color green
        ]
    ]
]
```

We see that the virus spreads through the network (Figure 1). Initially, three agents are infected, and within a few time steps, we see the red color starts spreading. If the probability of becoming resistant is high, the disease stops spreading early compared to a lower level of recovery-chance. Figure 2 shows that with a low value of recovery-chance, the virus spreads to about 40% of the population since agents get sick again after recovery (becoming susceptible), while it is contained to below 10% with recovery-chance equal to 10%.

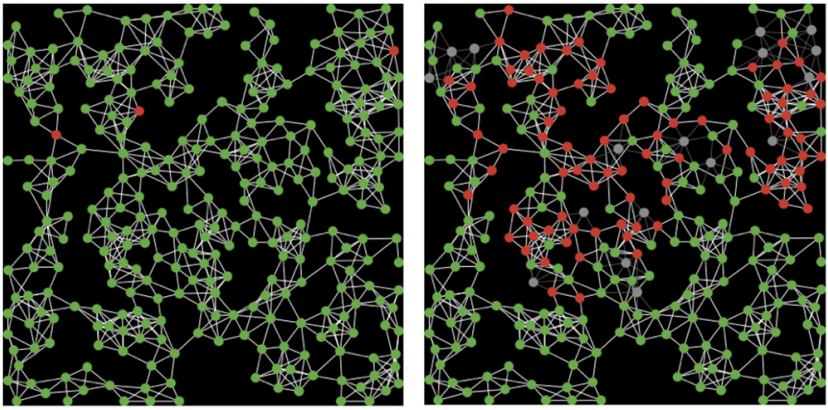

Figure 1. A network of agents where green agents are susceptible, red agents are infected and grey agents are recovered agents. On the left is the initial state of the network. Right is the network after 214 time-steps. The probabilities used are virus-spread-chance = 0.025, recovery-chance = 0.05 and gain-resistance-chance = 0.05.

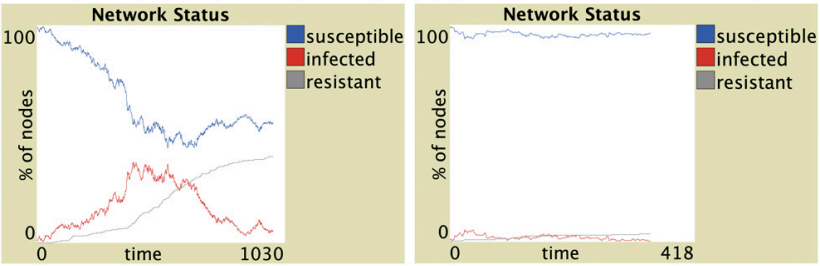

Figure 2. Spread of the virus in the population with recovery-chance = 0.05 (top) and recovery-chance = 0.5.

We will now explore the consequences of the parameters virus-spread-chance and recovery-chance and assume agents who recover do not become susceptible again (gain-resistance-chance = 1). We will see the effects for average degrees 4, 6, and 9 for 500 agents. Each parameter combination is run 100 times. Figure 3 shows that a higher degree leads to more agents getting the virus. Furthermore, we see that a higher probability of spreading the virus leads to more agents getting sick, and a lower recovery rate also leads to more agents getting sick.

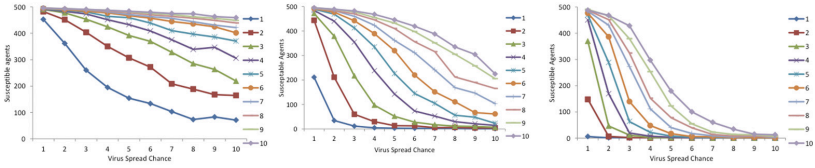

Figure 3. The average number of susceptible agents out of a population of 500 after 100 runs of 1000 time steps for different parameter settings. The upper figure is for an average degree of 4, the middle for an average degree of 6, and the lower figure for an average degree of 9. The horizontal axis shows the probability the virus is spread (from 1% to 10%) and the individual lines are the different levels of recovery chances (1% to 10%).

Policy Intervention

In order to stop the spread of infectious diseases, we have a

number of options. We can vaccinate people if a vaccine is available. With vaccination, we reduce the level of susceptible individuals and will reduce the value of R0. Another option is to reduce the interactions between agents. In practice, schools are often closed during an outbreak of a virus since children are an effective host in the spread of diseases like the flu. With sexually transmitted diseases like HIV, targeting prostitutes is a common approach since they are the hubs in interactions between agents. As an exercise, we can implement the option of deactivating an agent by clicking them with your mouse. Run the model and see whether you can isolate the virus (Figure 4). As you can see, this is not an easy approach.

To implement to mouse option, you need to include some code like

```
if mouse-down? [
   ask turtles [
      if distance patch mouse-xcor mouse-ycor < 1 [
         set infected? false
         set resistant true
         set color blue
         ask my-links [set color blue - 2]
      ]
   ]
]
```

You may have to slow down the speed of the model (slider in Interface) to avoid the model going too fast compared to your ability to click on the turtles you like to isolate.

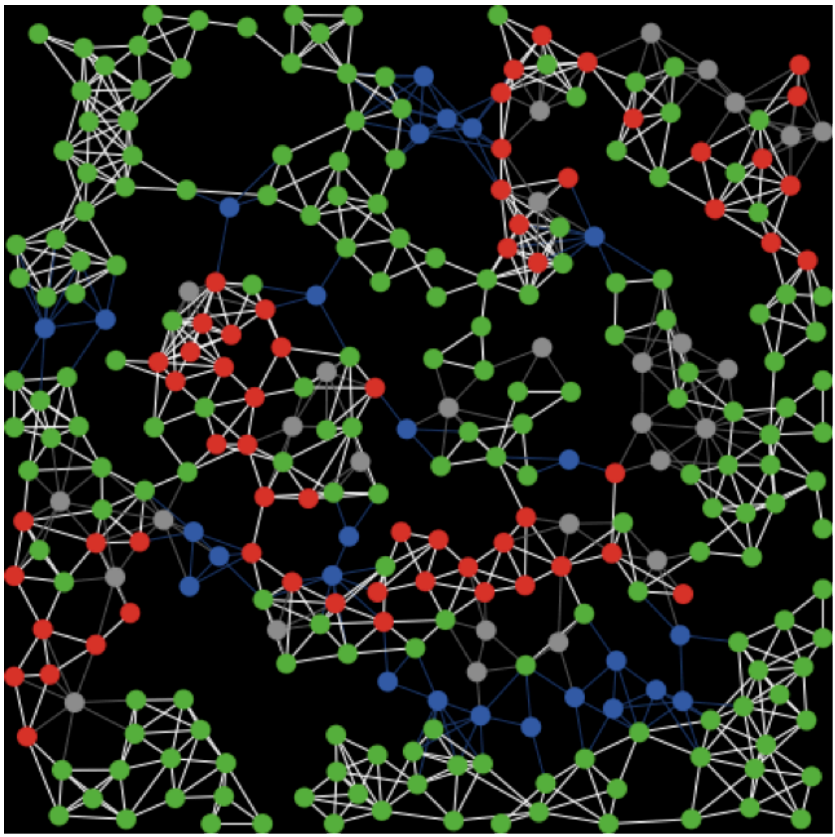

Figure 4. Screenshot of a network where the user tried to isolate virus (red nodes) by deactivating nodes (blue).

Another approach is to deactivate the links with the highest degrees. To model this in NetLogo, we create a while loop in the setup procedure, and we find one of the turtles that have the most neighbors and have not been isolated yet. We then isolate that agent. When an agent is isolated, we give them the color blue.

```
let i 0
while [i < numberofisolatedagents]
[
  let target max-one-of turtles with [color != blue] [count
```

```
  ask target [
    set infected? false
    set resistance? true
    set color blue
  ]
  set i i + 1
]
```

We simulate the model 1000 times for different numbers of agents to be isolated, and we see that more agents are not getting sick. What is the benefit of isolating a person? In Figure 5, we show that for each isolated person, on average, one or two other persons are spared. However, after isolating 80 agents, the marginal benefit starts to decline.

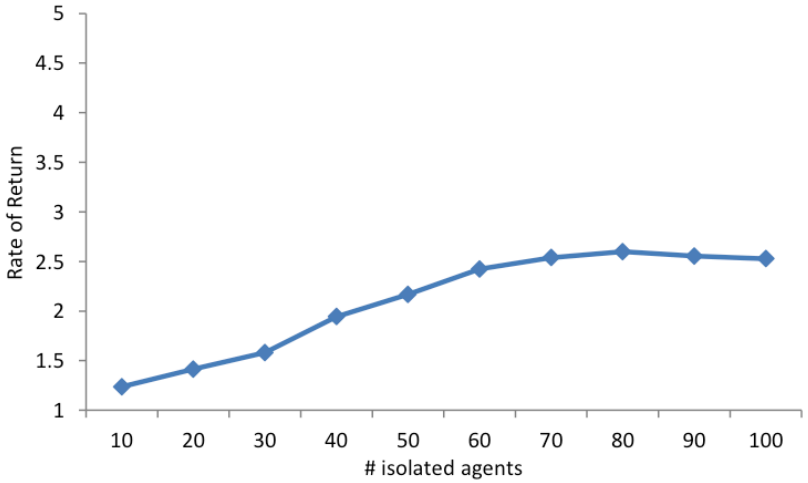

Figure 5. The rate of return (reduced number of infections – isolated agents, divided by the number of isolated agents).

Looking at an example of what the strategy is doing (Figure 6), we see that isolating the agents with the highest degree levels is unnecessary since it is, in effect, isolating agents far away from the infection. Furthermore, it does not isolate the spread of the disease.

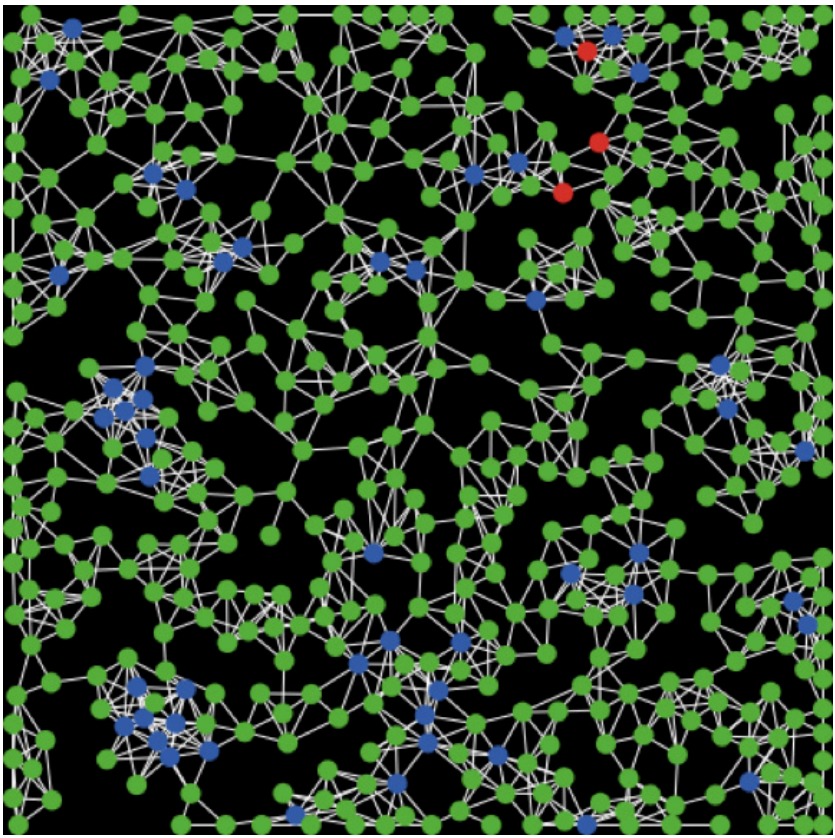

Figure 6. Screenshot of a network with 50 isolated agents.

Of course, it would be the most effective strategy to isolate the nodes next to the infected agents, but that would not be a fair strategy since there is incomplete and delayed information in reality about who is infected.

Suppose that we can isolate one agent each tick, which agents would we want to isolate? We can choose one agent with the highest degree each tick within a radius X of infected turtles. A smaller radius would indicate that the controller who is isolating agents had better information on where infected agents are. Nor surprisingly, we see that a short distance leads to better

performance (Figure 7). But not too close, however, since there might not be a sufficient number of highly connected nodes in a small radius that could stop the spread. Figure 7 also shows that more than 50 isolated agents start giving less effect per isolated agent. As an exercise, try to implement this strategy and see whether you can replicate the results in Figure 7.

Although agents are not isolated at the start of the simulation, the performance is much better than isolating agents with a high degree at the start of the simulation because this adaptive strategy is aimed at stopping the spread of the disease.

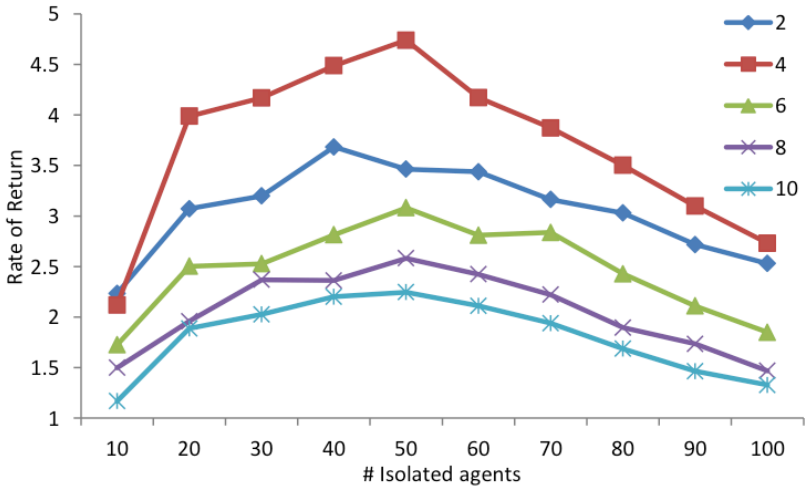

Figure 7. Average rate of return in isolating agents when different radius distances and a different number of isolated agents are used.

COVID-19

In 2020, the coronavirus SARS-CoV-2 caused a global pandemic. Many models are in development to explore the consequences of different strategies such as physical distancing, wearing masks, washing hands, lockdowns of cities and states, etc. Since we are during the COVID-19 crisis at the time of writing, and the scientific insights change rapidly, we do not present a specific model here.

There are a few differences with a simple SIR model. First, due to the asymptomatic spread of the virus, one could expect different infection rates if somebody is asymptomatic versus being asymptomatic and thus feeling sick, causing the agent to isolate itself from others. Another issue is the testing for COVID-19. Agents can be tested positive, correctly or not, which leads to getting into quarantine. Since it takes time to get test results, one may explore the consequences of delays in getting test results. A third component of a COVID-19 model will be the hospital capacity. If hospitals get near capacity, the death-rate will increase, which is one of the main reasons to flatten the curve. A fourth item of the coronavirus pandemic, which is interesting to explore with a model, is the political connection with wearing masks. Agents socially influence each other on their opinions to wear masks and comply with recommended behavior. Due to the weeks of delay between infection and visible consequences of a local outbreak, social influence plays an important part in the attempts to change behavior and reduce the reproduction rate. It also has to be acknowledged that various parts of the disease cycle are unknown, such as, are recovered patients immune to the virus or can they become susceptible again?; will there be an effective vaccine?; what is the infection rate in different kind of locations, and between different age-groups?

Summary

The spread of diseases can be modeled using the SIR model. In this chapter, we demonstrated how the SIR model could be implemented with agents on a network. The links relate to connections between agents that could lead to the spread of infectious disease. These connections could include being colleagues, commuters on the same bus, or sexual partners. We also demonstrated that network-oriented policies could be used to stop the spread of the disease effectively.

CHAPTER 19

Diffusion of Innovations

Key Concepts

In this chapter, you will

- study how innovation spreads in populations,
- be presented with models to study diffusion,
- explore the effect of network structure on the diffusion of innovation.

Facebook was introduced in February 2004 by Mark Zuckerberg from his dorm room at Harvard University. In March 2020, 2.6 billion people had a Facebook account. This is a dramatic success story of the diffusion of an innovation. Usually, when somebody develops a new product, it can take years to get it out on the market. For example, the Sony Betamax was considered to be a technologically superior product for recording TV programs, but it never became popular. Instead, the VHS system from JVC became the dominant format. There are many flops and some successes in innovations. In this chapter, we will use models to explore why this is the case.

The study of diffusion focuses on the attributes of innovations

that affect the speed and scope of diffusion. The social network structure can affect the spread. When the average path length of agents within a network is small, the innovation may spread faster. However, the diffusion speed and scope also depends on the medium of diffusion. Is an innovation easy to acquire when one comes in contact with it? Can the idea be spread via mass media, or does one need to train a person to acquire the skill? In this chapter, we discuss a number of examples of diffusion processes. We will look at the diffusion of innovations and the adoption of new products.

Diffusion of Innovations

The diffusion paradigm was developed by Ryan and Gross (1943), two rural sociologists who studied the diffusion of hybrid corn seed in two Iowa communities. The new hybrid corn, introduced in 1928, increased production by 20% and was more drought-resistant. However, farmers needed to purchase new seeds each year, instead of self-selecting seeds from their own plants. The new technology, therefore, led to significant changes for the farmers. By surveying more than 300 farmers in 2 communities during 1941, Ryan and Gross showed that diffusion is a social process that spreads adoption in the community through subjective evaluation, rather than individual rational choice.

Diffusion processes often replicate the observed stylized fact of an S-shaped curve of cumulated adopters of the innovation. In fact, the increasing number of adopters is, in essence, the diffusion process. The adoption of new products is a complex process that typically consists of a large body of agents interacting with each other over a long period of time. Traditional analytical models described diffusion processes on the market level since available data is often aggregated without attention to spatial distribution or individual characteristics. When individual data is available, these are snapshots of adopters' surveys.

Diffusion processes have been studied in many disciplines, such as anthropology, marketing, rural sociology, and communication

science. Everett Rogers (1962) synthesized the work of many studies and developed a framework of typical successful innovations. Innovators make the initial adoption of an innovation, after which early adopters, early majority, late majority, and finally, laggards may adopt the innovation.

As we develop a model of the diffusion of innovation, we need to consider that there is variability within the population when adopting something new (i.e., innovators versus laggards). Furthermore, there is likely interaction among agents that leads agents to imitate or copy innovators' behavior. The position of the innovator within a social network may affect the diffusion pattern.

Although the empirical investigation of dynamic micro-level processes is limited and difficult to accomplish, computational models have been developed, which can be used to investigate the consequences of different assumptions of agents on the diffusion of an innovation.

The traditional model on the aggregate penetration of new products is the Bass model (1969), which follows Rogers' diffusion of innovation (Rogers, 1962). The Bass model emphasizes the role of communication, namely external influence via advertising and mass media, and internal influence via word of mouth. An individual's decision is described as a probability of adopting a new product at time t, which is assumed to depend linearly on two forces. The first force is not related to previous adopters and represents the external influence, and the other force represents the internal influence and is related to the number of adopters.

$N[t+1] = p * (m - N[t]) + (q/m) * N[t] * (m - N[t])$

where $N[t]$ is the cumulated number of adopters at time t, m is the potential number of adopters, p reflects the influence that is independent of the previous adoption, and q reflects the influence due to word of mouth. Although the Bass models fit very well with much of the data, its relevance to real consumer behavior is questioned. One of the problems is that the model only explains an

observed successful diffusion process, and cannot predict whether a new product will be a success or a failure.

One way to simulate the diffusion process from an agent-based perspective is to equip the agents with thresholds when to adopt an innovation. For example, Mark Granovetter (1978) presented a framework that assumed that thresholds to join a collective behavior differ among individuals and that the type of distribution affects the system's macro-dynamics. Granovetter (1978) focused on processes in which the whole group could monitor adoption. More relevant for many diffusion processes, is to base the threshold model on personal networks. Valente (1995) analyzes in detail threshold models for a number of data sets. Thresholds in his models are based on the individual personal network. Interestingly, less risky innovations spread faster if the central connected agents have a low threshold for adoption. For more risky innovations, less connected individuals adopt the product first, which may reduce uncertainties and lead to diffusion to the whole group afterward.

Simple Diffusion Model

We will now present a series of NetLogo models of diffusion processes which can be downloaded here. In the first model, we have agents walking randomly around. All agents are randomly placed in the environment. With a probability probstep, an agent adjusts its direction and will move one step forward. All agents are white except for one who has the color red. Every time-step, an agent will check the agents' fraction with the red color in the patch the agent is located. If the fraction of red agents is above the threshold "threshold" of the agent, a white agent will become a red agent. We will compare each agent's effect with a threshold equal to 0.5, with each agent having a threshold value randomly drawn between 0 and 1. To implement this in NetLogo, we use the code below. Turtles-here is the set of all turtles on the caller's patch. By adding the other set will not contain the caller. It checks first whether sum0 + sum1 is larger than zero to avoid division by zero.

If the fraction of red turtles is higher than the threshold, the color of the agent will be red and will stay the same, white or red, otherwise.

```
let sum0 0
let sum1 0
ask turtles
[
  set sum0 count other turtles-here with [ color = white ]
  set sum1 count other turtles-here with [ color = red ]
  if sum0 + sum1 > 0
  [
    if (sum1 / (sum0 + sum1) > threshold)
    [
      set color red
    ]
  ]
]
```

With an initial density of 1% of red agents, we see a slowly growing number of red agents (Figure 1). When the threshold values vary among the agents, the diffusion is faster than the same threshold value for all the agents. This is illustrated in Figure 2, where the heterogeneous case has a smaller number of ticks to come to the diffusion of red agents compared to a homogenous threshold of 0.5. Some innovators are more likely to adopt the innovation when there is heterogeneity when just one red agent is on the patch.

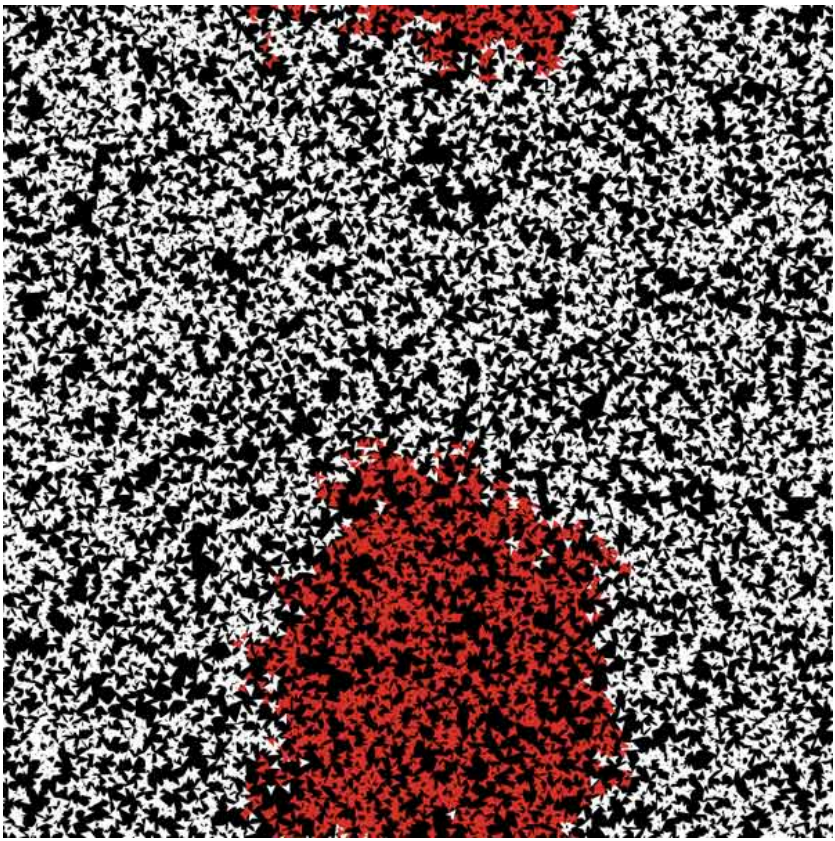

Figure 1. The spatial pattern of 10000 agents.

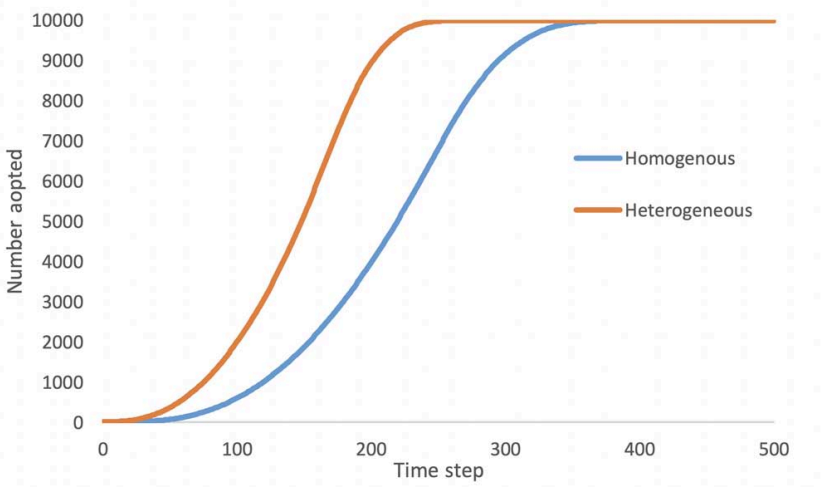

Figure 2. The number of adopted agents for heterogeneous threshold values (orange) and a fixed threshold value of 0.5 (blue) for all the agents.

Diffusion in a Social Network

Suppose agents are connected within a social network, like a small-world network. Starting with one red agent, white agents who have a red neighbor will adopt the innovation. This leads to the spread of innovation through the network. As you can see in Figure 3, the diffusion takes short cuts of relinked nodes. The shorter the average path length in a network, the faster the innovation will spread.

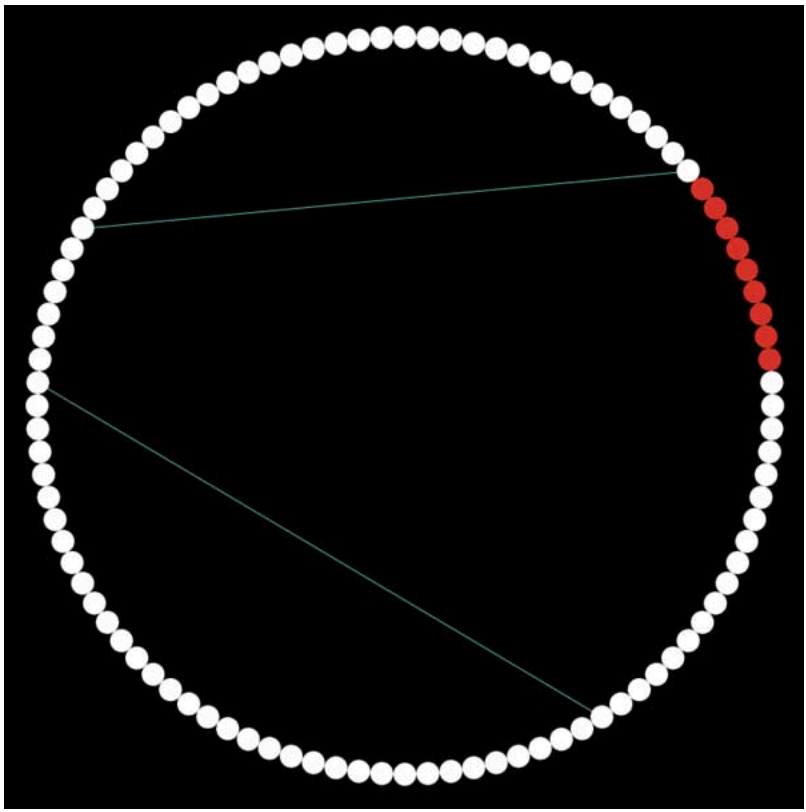

Figure 3. Snapshot of the spread of innovation on a small-world network.

To model the diffusion in networks, we count for each node the number of friends that are red. All agents are updated synchronously. Therefore we first define the number of red friends for each node, nan, and then check whether the threshold is passed. Check out what happens if the updating is not synchronous. The basic code of the model is now as follows.

```
ask turtles [
  let node1 self
  set nan count turtles with [link-neighbor? node1 and colo
]
```

```
ask turtles [
  if nan >= threshold [ set color red]
]
```

In Figure 4, some simulation runs are provided for a network of 500 nodes with different rewiring probabilities. There is one red agent at the start of the simulation. Agents adopt when at least one of their neighbors has adopted. We see that when all agents are in a circle—-no rewiring (=0)—-the number of red agents spread linearly until all agents are adopted. When there is a small probability of rewiring, the adoption is also linear until the short-cut is hit, which speeds up the adoption rate. We see that for larger rewiring rates, the diffusion goes faster, where a rewiring rate equal to 1 leads to a random network.

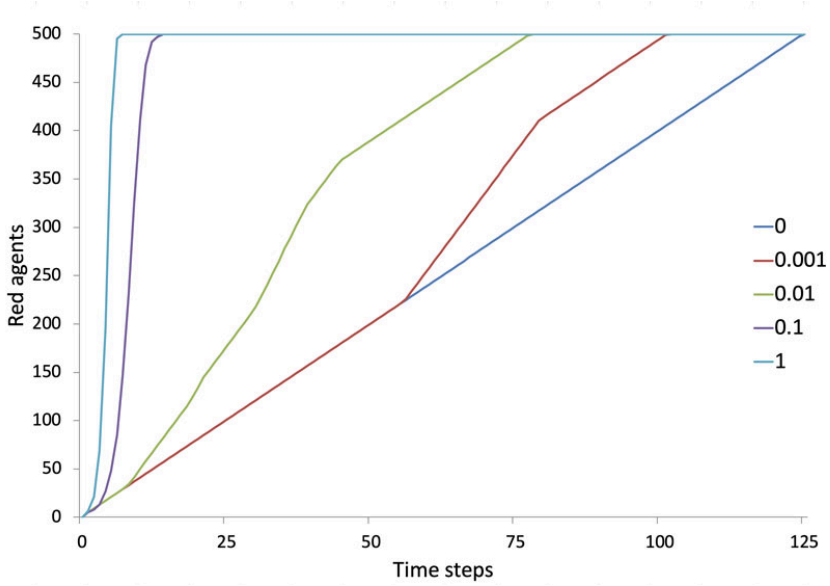

Figure 4. Diffusion in networks with different rewiring probabilities.

If we change the threshold to two neighbors needing to have adopted the innovation before the agent adopts, we see that the adoption will not spread through the whole population when there

are rewiring of the original regular links between neighbors. Figure 5 shows the number of adopted agents out of 500 agents after 500 tickets, averaged over 100 runs. When there are more random links, there is a smaller probability the adoption spreads far. With more random links, the clustering coefficient is reduced, and it becomes less likely that at least two neighbors have adopted the innovation.

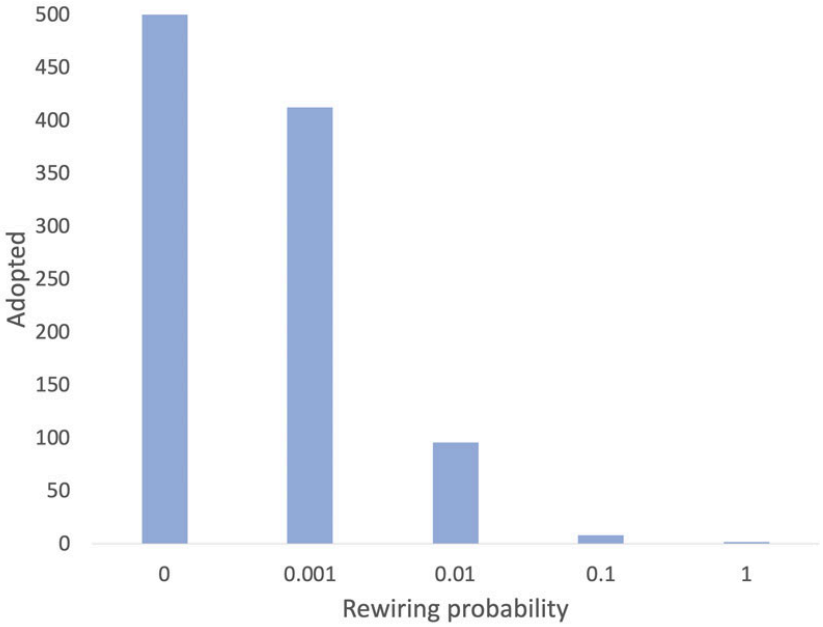

Figure 5. The number of adopted agents for different probabilities of rewiring.

Lock-in of Products

Why do we sometimes have one product dominating a market, even if it is a product that some argue to be inferior? Examples are Microsoft, QWERTY keyboards, the Google search engine, etc. This is the problem of lock-in. Economist Brian Arthur (1994) studied the problem of lock-in of technologies and developed some models that simulate lock-in processes.

The basic idea is the following. Suppose there are two products to chose from, say Microsoft and Apple. The more people in your

environment with whom you share documents who use product A, the more valuable product A is. If a person has to choose between two products, he or she will be more likely to choose the product with a majority of users, if the two products have similar performance.

We can implement this process in a simple model. Every time step agents made a choice. The choice for product A is proportional to the share of product A of both products as chosen in the previous time step. Starting the model with equal probabilities, one of the two products starts to dominate. Which product will dominate cannot be predicted from the start. Random events may give one of the products a larger market share and stimulating dominance in future time steps (Figure 6).

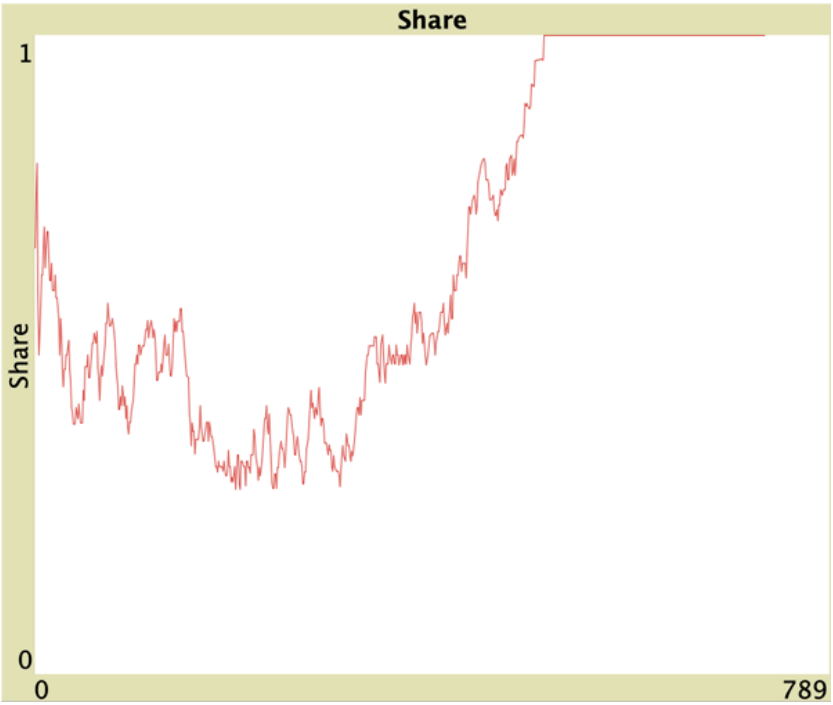

Figure 6. Share of red products among agents who bought a product.

In Figure 6, we show results of a model where every time-step, another agent will decide to enter to make a decision between products A and B. Every time step agents who have made a choice between A and B will reconsider it based on the current adoption shares. Suppose agents will not reconsider their decisions, then we may get stable mixed shares of products A and B (Figure 7). Once a large number of agents have made their choice between A and B, every new agent will choose relative to the frequency of products A and B. Hence, we need to include some reconsideration of agents in order to derive lock-in of one of the products with certainty.

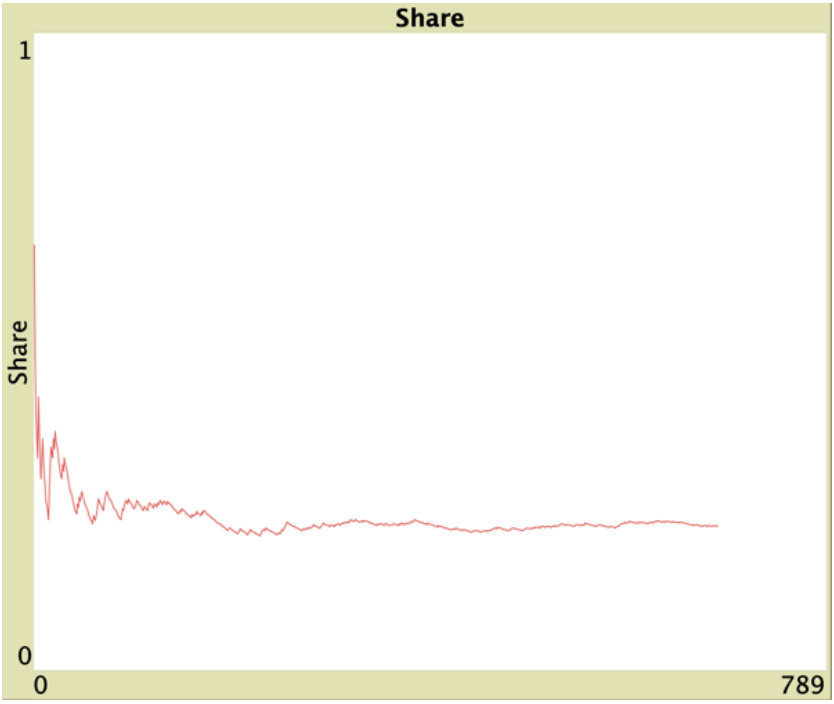

Figure 7. Share of red products among agents who bought a product when agents do not reconsider.

In Figure 8, you see the results of a large number of simulations. We run the model for 1000 time steps, and use different probabilities

for agents to reconsider their choices. For each probability, 1000 simulations were performed, and the fraction of product A was determined at the end of the simulation. Then we ordered the simulations according to the share of product A. We see that for a reconsideration probability of 0, the share of product A follows a uniform distribution. However, for probability equal to 0.02, one-third of the simulations have a fraction of zero, and one third has a fraction of 1. Hence the distribution is bipolar.

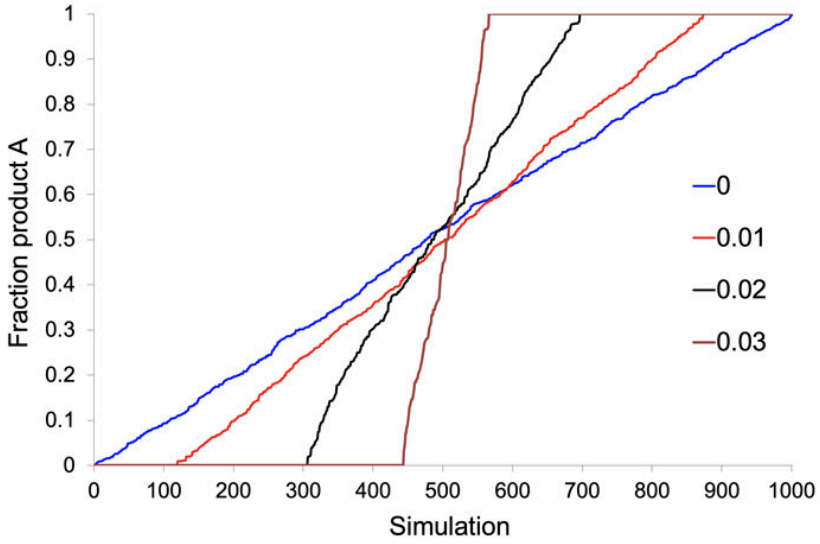

Figure 8. The fraction of shares of product A at the end of 1000 simulations for different probabilities of agents reconsidering their choice.

Summary

How do new technologies spread within a network of consumers? In this chapter, we discussed various classic models to capture the traditional S curve of diffusion of successful innovation, and the lock-in of technology at the cost of a competing technology. Typical rules of adoption relate to the share of products within a (local) network and the thresholds that stimulate agents to adopt

an innovation. In the next chapter, we extended this work by including more explicit different types of motivations of the consumers.

References

Arthur, W. Brian (1994) Increasing returns and path dependence in the economy. Ann Arbor: University of Michigan Press.

Axelrod, Robert (1997) Dissemination of culture: A model of local convergence and global polarization. *Journal of Conflict Resolution* 41(2): 203-226.

Bass, Frank M. (1969) A new product growth model for consumer durables. *Management Science* 15: 215-227.

Gladwell, Malcolm (2000) The tipping point. New York: Little Brown.

Granovetter, Mark (1978) Threshold models of collective behavior. *American Journal of Sociology* 83: 1420-1443.

Rogers, Everett M. (1962) The diffusion of innovations. New York: Free Press.

Ryan, Bryce, and Neal C. Gross (1943) The diffusion of hybrid seed corn in two Iowa communities. *Rural Sociology* 8: 15-24.

Valente, Thomas W. (1995) Network models of the diffusion of innovations. Cresskill, NJ: Hampton Press, Inc.

CHAPTER 20

Fads and Fashions

Key Concepts

In this chapter, you will

- be introduced to a model of consumer behavior,
- experience agents who use different heuristics dependent on if they are satisfied or uncertain,
- explore the consequences of different assumptions of decision making on market dynamics.

What is in style? Many people follow trends in the clothes they wear, their haircuts, the kind of TV shows that they watch, the kind of toys children play with, etc. And what is in style changes over time. What is clear is that people like to imitate the behavior of others. Of course, some people do not want to join the majority, creating a counter-culture.

What affects fads and fashion in our behaviors? Why are some items, like clothing, more fashionable than others, like detergent? Why do fashionable items disappear over time?

These are the questions we will address in this chapter, mainly focused on how we, as consumers, make decisions. The modeling

approach presented in this chapter is based on the work that the author did with social psychologist Wander Jager. In the references, we refer to some relevant publications for further study.

Consumer Behavior

In the previous chapter, the agents adopted products because a number of other agents used that product. The agents did not evaluate the product itself. In a more realistic model of product choice, agents may make decisions based on different attributes of the products affecting different needs of the agents. In this chapter, we will include more psychology in modeling the decision making of the agents.

We assume that agents try to satisfy their needs. One activity, such as eating a meal with your family, can satisfy the needs of subsistence as well as the needs of belongingness. When people make decisions, they take into account the different types of needs that are involved. In this chapter, we simplify the model to include an individual need and a social need. The individual need refers to the functionality of the product; it is a product that the agent likes. The social need refers to the fact that people want to belong to a group and like to conform to the choices made by their friends.

There are different needs, and various scholars have tried to organize the diversity of needs. Maslow's hierarchy of needs is the most prominent organization of needs. Abraham Maslow argued that individuals initially try to satisfy physiological needs that are needed for human survival and can be satisfied by eating, drinking, and having sex. The second level of needs is safety needs, which require financial security, health, and a safe environment. When those needs are satisfied, individuals will focus on higher needs, such as social belonging (friendships, family, social groups), self-esteem, and self-actualization.

Another classification was the list of fundamental human needs proposed by Manfred Max-Neef. Max-Neef does not define a hierarchy but list nine essential needs, namely: subsistence,

protection, affection, understanding, participation, leisure, creation, identity, and freedom.

This spectrum of different needs is in contrast to the abstract notion of utility used in neoclassical economics as a preference ordering over a choice set, ignoring the potential conflicts of diverse sets of needs. In the model in this chapter, we include different types of needs, but not the spectrum psychologists proposed. As modelers, we have difficulty quantifying the dynamics of need satisfaction and trying to keep things simple. More sophistication of modeling needs and choice is an important agenda item for modelers.

A Model of Consumer Behavior

We will present a simple model in which agents make decisions on which product to buy from a set of N possible products. The model can be downloaded here. Agents are connected in a social network. Agents buy products according to the decision rules and evaluate their choices relative to what happens in the neighborhood of their social network.

The utility of using a product consists of an individual part and a social effect part. The individual part expresses the degree of fit between the product characteristics and the preferences of the consumer. In the model, this individual utility is expressed as the difference between the single personal preference of a consumer and the single product characteristic. The personal preference of actor i, p_i, is expressed by a value between 0 and 1. The utility of the product, based on personal preferences alone, is equal to one minus the absolute difference between personal preferencep_i and product dimension δ_j. Hence the smaller the difference between the individual preference and the product characteristic, the closer the individual utility gets to its maximum value of 1.

The social effect means that the utility of a product increases when more friends consume the same product. This effect only affects social needs satisfaction. Hence, this effect qualitatively differs from the positive network effect often discussed in markets

such as operating systems and online games and the like, because the individual utility is not affected by the number of friends consuming the same product. This social effect relates to the need to conform to others in the agent's network. The variable denotes, therefore, the average distance of choices made by the agents and their neighbors. For each agent, we calculate the ratio of products as a fraction of all products consumed among the agent's neighbors. The total expected utility of consuming product j is equal to:

$E[U_{ij}] = \beta_i * (1 - | \delta_j - p_i |) + (1 - \beta_i) * x_j$

The components of the utility function, the individual part, and the social part are weighted with β_i and $1 - \beta_i$, with $\beta_i \in [0,1]$. A low β_i holds that personal need is weighted less, as is usually the case with fewer people who are less focused on innovations (Rogers, 1995). In contrast, a high β_i holds that social needs are weighted less, as is usually the case with people who are more focused on innovations. The dimension δ_j on which the agents make decisions can represent various product characteristics.

The agents are located in a network. We first place agents randomly on the landscape and then we link agents to both nearby, and faraway agents (Read and Keeling, 2003). The probability of any two agents being connected is calculated so that the probability of a connection between any two agents decreases as the spatial distance between them increases. The probability that a short-length connection will be made rather than a long-distance connection is determined by a parameter, D, which represents the desired link length in the network.

Given two agents separated by distance d, the probability of a link connecting them is:

$p = (n / (2 * \pi * D^2)) * e^{-N * d * d / (2 * D * D)}$

where n is the average number of links in the network, d is the distance between two agents, D is the desired length-scale for generating networks, and N the number of agents. When D is small, i.e., D = 2, agents will be preferentially connected to agents

within their immediate spatial vicinity. When D is larger, i.e., D = 10, agents are more connected with individuals at greater distances from themselves, that is, they are more "globally" connected (Figures 1 and 2).

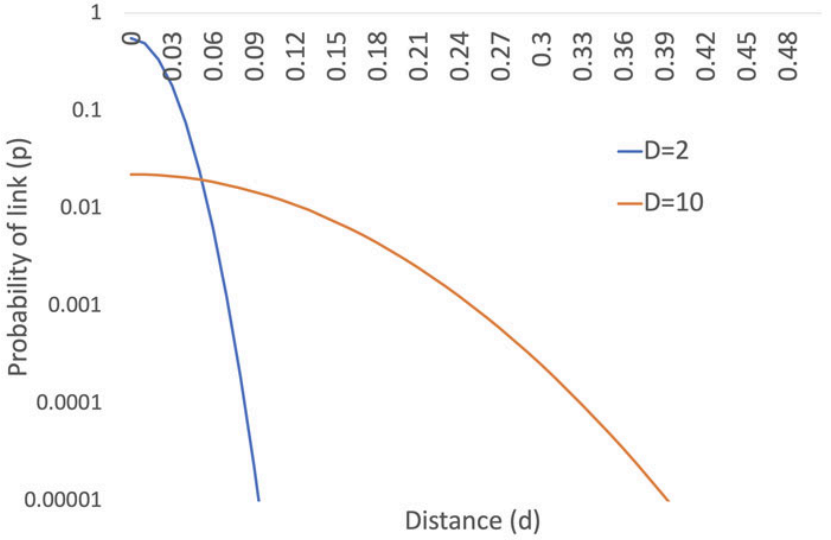

Figure 1. The effect of D on the network structure. With D=2 the connections are more local compared to D=10. Parameter n = 14, and N = 10,000.

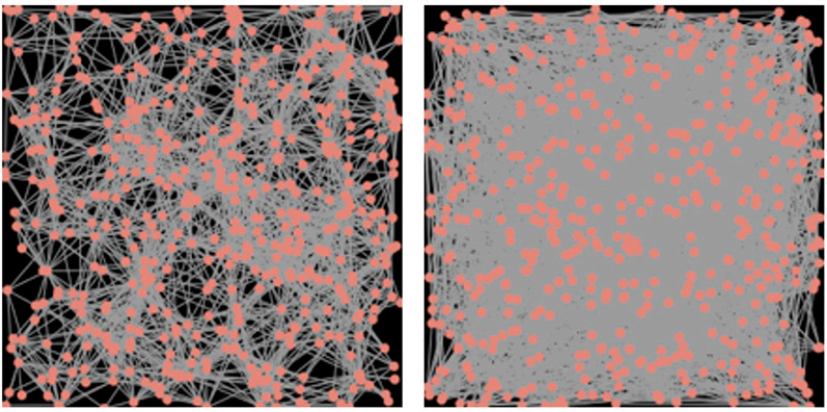

Figure 2. Network of 500 agents using D= 2 (top) and D = 10 (bottom).

Each agent makes a decision to buy a product with probabilityp_b (=0.10). When an agent makes a decision it will continue the choice of past rounds if the experienced utility is higher than the minimum utility that satisfies the agent u_{min}. If the agent is not satisfied, the agent will calculate the expected utility for all N products and chooses the one which will maximize her utility. Since agents will use less cognitive energy if they repeat decisions from the past, it is common to repeat past decisions if you have been happy with the outcome. Only when it is needed will you evaluate all the options.

We now turn to some exercises. Let's assume all agents have the same values for β and u_{min}. We will vary β and u_{min} and run the model 100 times for each parameter combination. All agents are initialized randomly with one of the available products.

What do we measure from the simulations? We would like to know whether the products are distributed evenly among the agents, which would be the case if they only care about their own individual preferences. Therefore we calculate the Gini coefficient of the products bought by the agents at the end of the simulation. Figure 3 shows a low Gini coefficient, the equal spread of the products among agents if agents weight the individual preferences

high, and when the minimum utility level is low. If β is 0.5 or higher, the Gini coefficient will rise above 0.5, especially if the minimum utility is high. In such situations, one or two products dominate the market. Agents value conformity and therefore buy similar products as their friends.

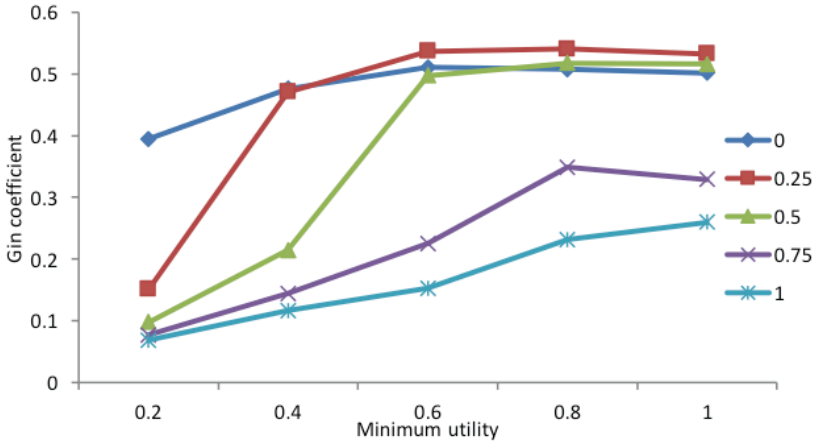

Figure 3. Gini coefficient of the distribution of products in the population for different parameter combinations (β and u_{min}). Results are averages for 100 runs of 100 ticks.

The other variable we measure is the average utility at the end of the simulation. In general higher utility is derived if the minimum utility is higher (Figure 4). The utility is also higher if they focus more on conformity. The reason for this is the relatively small number of choices, 10 products, and the uniform spread of the preferences. Products cannot fully satisfy individual preferences, but if people weigh social preferences more, they can get a high utility by buying the same products as their friends do.

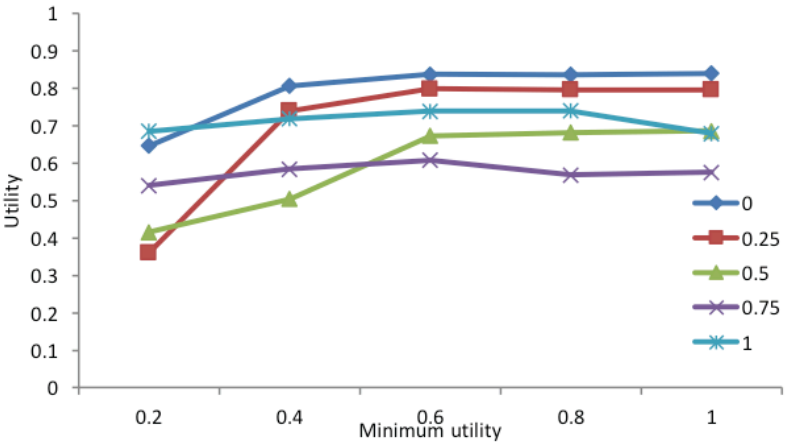

Figure 4. Average utility for different parameter combinations.

In the previous simulations, the β value of all agents was equal. What would be the consequence of agents having different values of β? It would mean that some agents are innovators, focusing on products meeting their individual preferences, while others are behind in adopting new products but care to conform with the majority in their circle of friends. We see that a heterogeneous distribution of β leads to a lower Gini coefficient (Figure 5) and lower utility (Figure 6)if the threshold of the minimum level of utility is high. As shown in Figure 3, a high β leads to a lower Gini coefficient, and the variation of β follows this trend.

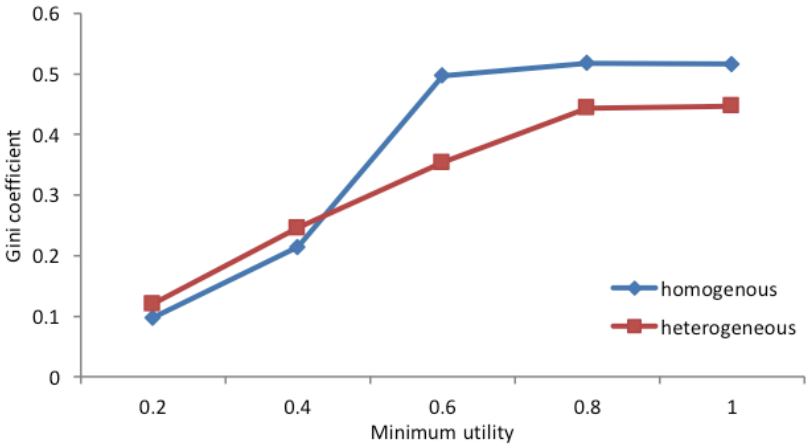

Figure 5. The Gini coefficient for different parameter values and where β was 0.5 or on average 0.5 but with a uniform distribution between 0 and 1.

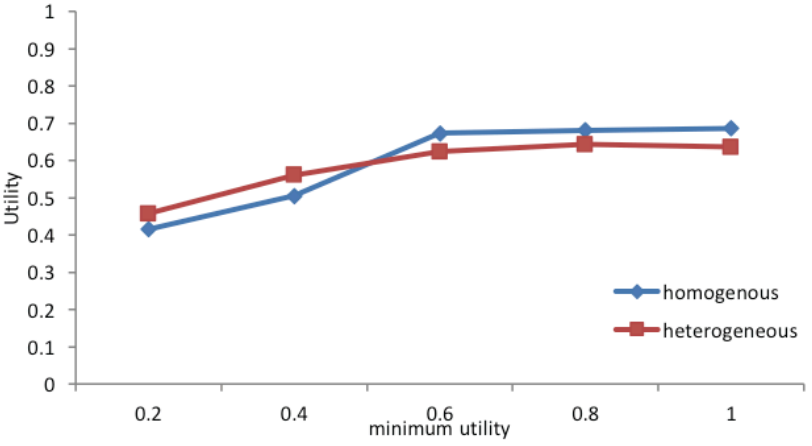

Figure 6. The average utility for different parameter values and where β was 0.5 or on average 0.5 but with a uniform distribution between 0 and 1.

Including Social Influence

In the model we described above, agents were repeating their decisions; or deliberating on finding the solution that maximizes

the expected utility. In practice, people often benefit from the experiences of other people in making a decision. Using the information on other people's behavior or experiences may function as a cue to simplify the decision task. Especially when people are uncertain (due to the complexity of the decision, uncertain personality) and the behavior in question is very visible (either by observation or discussion), social heuristics are used.

People are assumed to imitate if they are uncertain. When agents imitate, they follow the majority of the people in the community. This does not necessarily lead to a product that satisfies the agent. For example, an agent may imitate their friends' product choice even though they have different preferences. Another approach is social comparison, where agents look at the products used by agents like them, and the agent will evaluate those options.

The expected uncertainty $E[Unc_{ij}]$ reflects how uncertain an agent i is about having made the best choice when choosing product j. The more friends in their social network consume other products, the more uncertain an agent is. Furthermore, the value of β_i determines how sensitive an agent is to not have the same choice as his or her friends. The more important the social need, (the lower the value of β_i), the more uncertain an agent becomes when his or her friends consume different products:

$E[Unc_{ij}] = (1 - \beta_{ij}) * (1 - x_j)$

with $(1 - x_j)$ represents the fraction of the friends of agent i who consume a different product than agent i. It is essential that agents start engaging in social processing when their uncertainty exceeds a critical level, the maximum uncertainty level.

Agents make a choice between what they really like and whether they select the same product as their neighbors. We specify the different heuristics they use.

Deliberate

The agent evaluates all the options available and calculates the expected utility if one consumes the product. Based on the

expectation, she will choose the product with the highest expected utility.

Repeat

The agent will repeat her choice from the previous time.

Imitate

The agent will copy the most popular product in her neighborhood.

Socially Compare

The agent will evaluate the most popular product in her neighborhood and choose this option if it increases her expected utility compared to sticking with her current choice.

Figure 7 shows the chosen products in the social network and illustrates the clustering of choices in parts of the network. Although initially agents randomly chose a product, within a few time-steps, we see clusters emerging. Figure 8 shows that at the beginning of the simulation, there is a lot of social comparison and imitation. When the agents start to settle on their choices, they continue making repetitive choices.

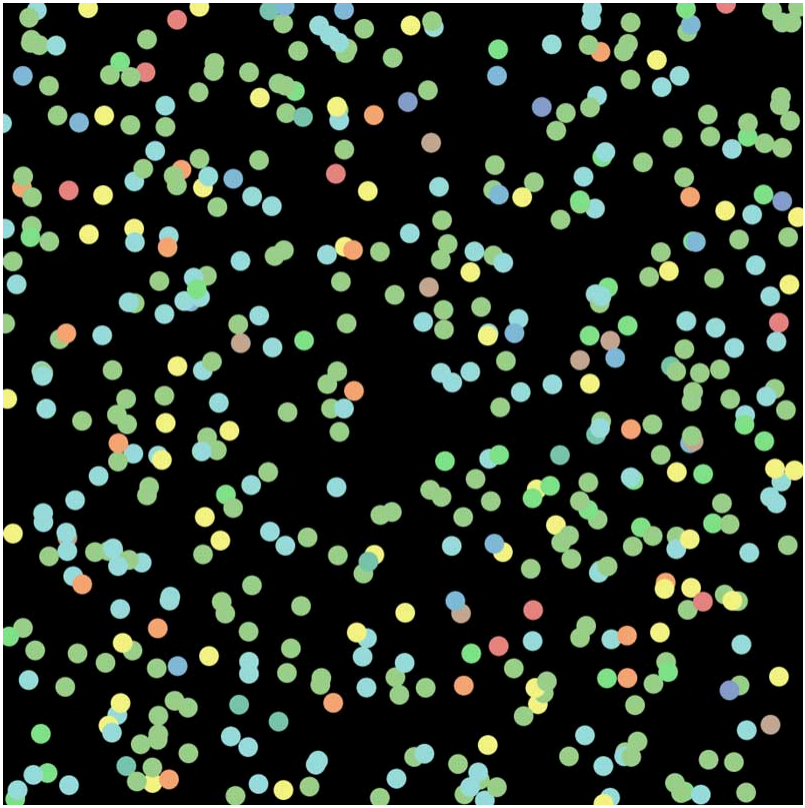

Figure 7. Population of agents (links hidden) where each agent is colored according to the product chosen.

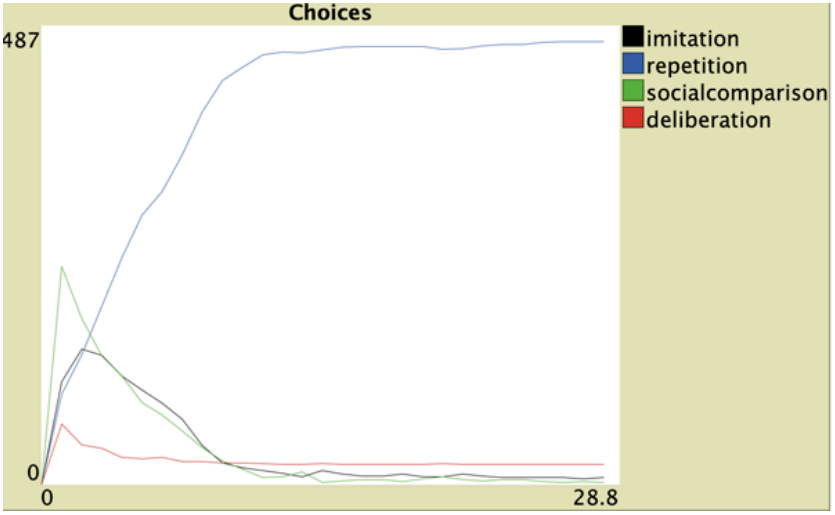

Figure 8. The number of agents who make choices according to a particular heuristic.

Next, we will explore the effect of different levels of the maximum uncertainty level when using a heterogeneous β and a minimum utility of 0.5. We ran the model 100 times 100 ticks at different levels of maximum uncertainty toleration. Figure 9 shows that when no uncertainty is tolerated, agents only imitate and socially compare. When the uncertainty tolerance is increased, we see repetition and deliberation being included. The Gini coefficient dropped and was highest when agents only imitated or socially compared (Figure 10). With a higher maximum allowable uncertainty, the average utility is increased a bit since imitation does not necessarily lead to solutions that have high utility.

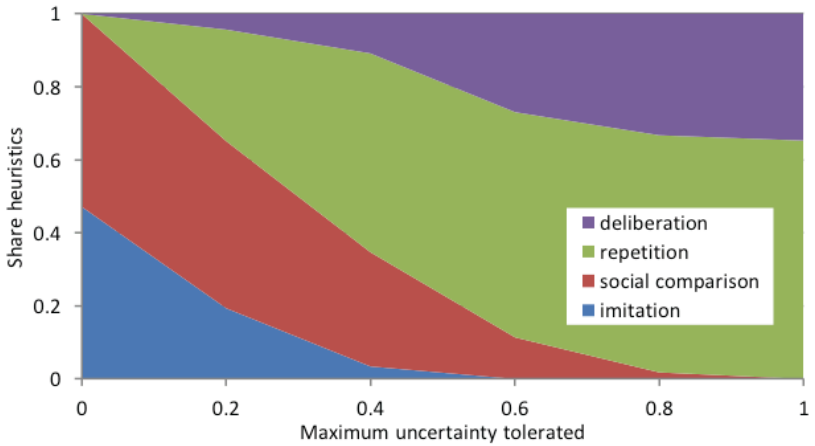

Figure 9. Distribution of heuristics for different levels of maximum uncertainty tolerated.

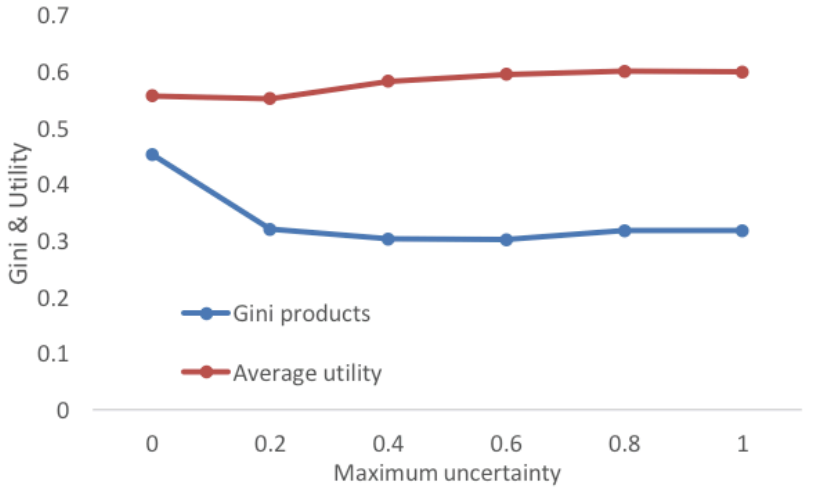

Figure 10. Gini coefficient of products and average utility for different levels of maximum uncertainty tolerated.

We will now explore the consequences if agents do not know all the available products at the start of the simulation. Suppose agents

have a memory of the products on the market. In previous simulations, we assumed that the agents had full knowledge about the opportunities. Suppose that initially, they have only a memory of products they and their neighbors are consuming. Agents build up their memory and thus their repertoire of possible products if they see one of their neighbors consuming a product.

Figures 11 and 12 shows that there is more equality if agents do not have full knowledge. Since not all options are known, and agents initially are randomly allocated a product, there remains a spread of many products in the population. When agents know the products available, we see that one or two products start to get a larger market share, especially when they use imitation.

The average utility is generally higher when agents have full knowledge. This is not surprising since they have more options available to increase their utility. However, when agents only care about what others do and imitate others' decisions, we see that there will be a higher level of satisfaction.

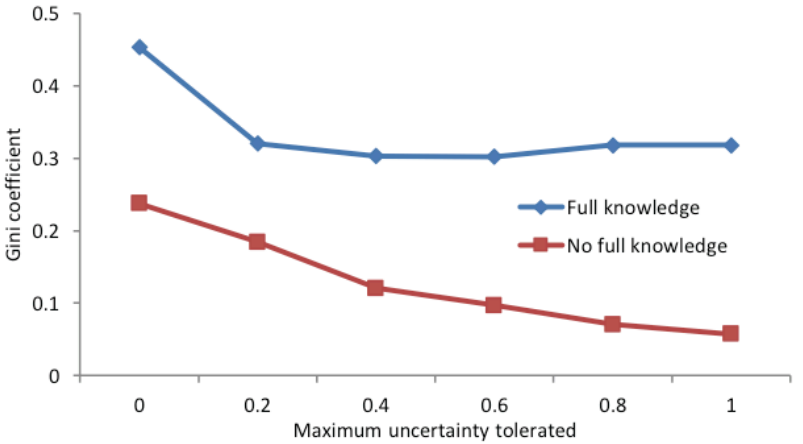

Figure 11. Gini coefficient for simulations with full and incomplete knowledge for different levels of the maximum tolerated uncertainty.

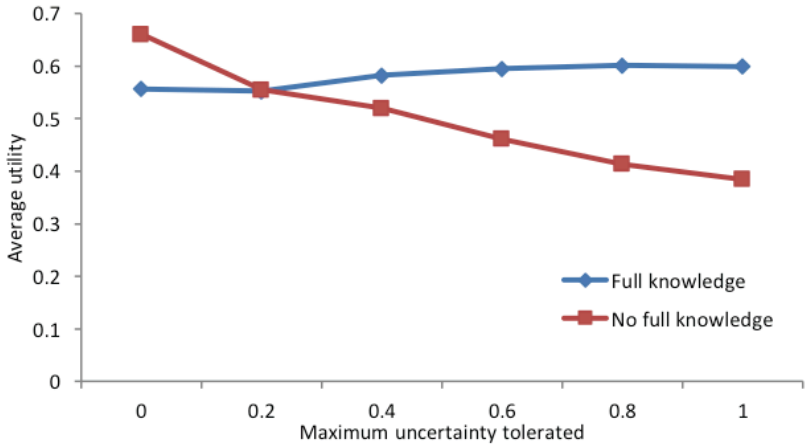

Figure 12. Average utility (bottom) for simulations with full and incomplete knowledge for different levels of the maximum tolerated uncertainty.

Figures 13 and 14 show that when agents have incomplete initial knowledge, they will use more deliberation and less repetition. Their utility is more frequently below the threshold of satisfaction leading agents to deliberate.

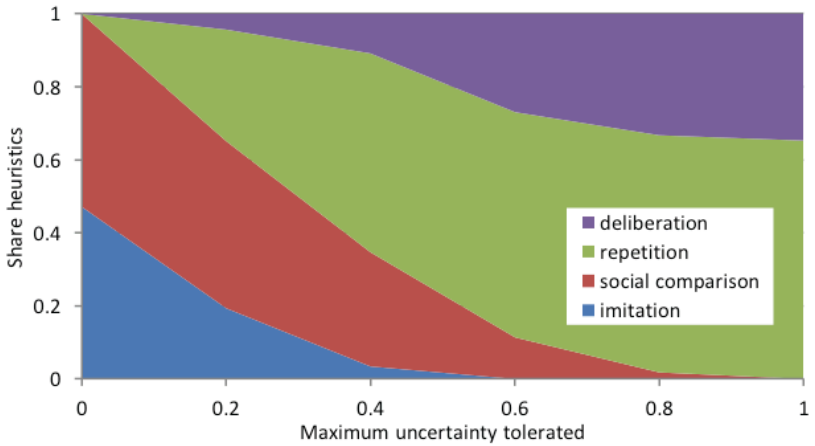

Figure 13. Share of different heuristics with full knowledge for different levels of the maximum tolerated uncertainty.

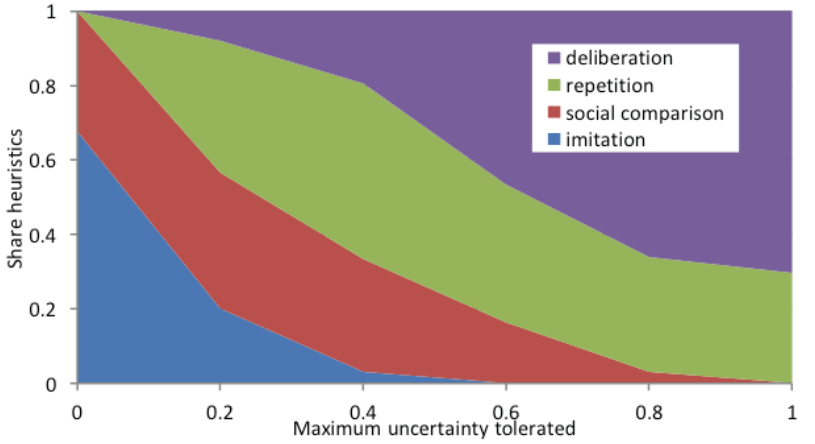

Figure 14. Share of different heuristics with incomplete knowledge for different levels of the maximum tolerated uncertainty.

Summary

In this chapter, we presented the consumat approach that defines agents using different decision heuristics in different types

of context. Since agents derive utility from the direct consumption of the product and whether their friends consume the product, a consumers' network is constantly changing since the local context of the consumers is changing. From the model analysis, we can infer how changes due to technology (social media) can impact the information have about others, and therefore also the market dynamics of consumer products.

References

Gladwell, Malcolm (2005). Blink: The power of thinking without thinking. New York: Little and Brown.

Janssen, Marco A. and Wander Jager (2000) Psychological factors affecting market dynamics: the role of uncertainty and need satisfaction, *Advances in Complex Systems* 3: 323-334

Janssen, Marco A. and Wander Jager (2001) Fashions, habits and changing preferences: a simulation of psychological factors affecting market dynamics, *Journal of Economic Psychology* 22: 745-772

Janssen, Marco A. and Wander Jager (2003) Simulating market dynamics: Interactions between consumer psychology and social networks. *Artificial Life* 9: 343-356.

Kahneman, Daniel (2011) Thinking Fast and Slow, MacMillan.

Levitt, Stephen D., and Stephen J. Dubner (2005) Freakonomics: A rogue economist explores the hidden side of everything. New York: William Morrow.

Maslow, Abraham (1954) Motivation and personality. New York, NY: Harper

Max -Neef, Manfred A. (1991) Human Scale Development: conception, application, and further reflections (New York, The Apex Press).

Read, Jonathan M, and Matt J. Keeling (2003) Disease evolution on networks: the role of contact structure. *Proceedings of the Royal Society B*, 270: 699–708

CHAPTER 21

Opinions and Politics

Key Concepts

In this chapter, you will

- be introduced to modeling opinion dynamics,
- learn about modeling adaptation of and competition between political parties
- explore combining models of opinion dynamics and adaptive parties.

People have opinions and those opinions can change when people interact with others. In recent years we have seen that social media creates echo chambers reinforcing opinion polarization (Iyengar et al., 2019) and the spread of misinformation (Del Vicario et al. 2016). There is substantial literature on opinion dynamics (Hegselmann and Krause, 2002; Noorazar, 2020) stimulated to explain the observations of polarization and propagation of misinformation.

The diversity of opinions can have an impact on social-ecological systems (Janssen and de Vries, 1998) since opinions on sustainability issues impact which political parties rule. There is

also substantial literature on voting dynamics where agents vote on issues or parties (e.g. Kollman et al., 1992; Laver, 2005) who look at the consequences of party performance and adjustment of party positions with populations who have fixed opinions.

We will first introduce a basic model of opinion dynamics, and then a model of party competition. Then we combine those models and look at the consequences of political dynamics if agents change their opinions.

Opinion dynamics

There are different ways to define opinions in a mathematical way. A key distinction is whether to consider a continuous or a discrete opinion space. Discrete opinion space is often based on Ising models which have a long history in statistical physics. We continue with continuous-space models and bounded confidence models (Hegselmann and Krause, 2002). The NetLogo model used can be downloaded here.

Each agent has an opinion and interacts with other agents. Agents influence each other. Agents with different opinions adjust their opinions to come closer to each other in their opinions. However, this only works if agents tolerate each other's opinions in order to have a fruitful interaction. If agents do not tolerate each other's opinion agents may not influence each other or even repel and get more distant opinions.

In more formal notation,

- an agent i has an opinion x_i,
- a threshold u which defines how big of a difference the agent still accept the opinion of the other agent j,
- and a threshold t which defines how big of a difference in opinions between agents i and j need to be to not tolerate the opinion of the other agent.

Agents randomly are paired and interact. If agents accept the

other opinion, they move towards each other. If the agents do not tolerate the opinion of the other agent, the opinions will become further apart.

The following rules are used for the adjustment of the opinion dimension *x*:

If $| x_i - x_j | < u$ change into $x_i = x_i + \mu \cdot (x_j - x_i)$

If $| x_i - x_j | > t$ change into $x_i = x_i + \mu \cdot (x_i - x_j)$

Where μ is the strength of the influence. The same rules apply for updating the opinion of individual j. The opinions are between [0;1] and we use the same threshold values of all agents. Initially, agents are randomly assigned values of the opinions.

We use a μ equal to 0.05 of all 1000 agents and vary the threshold values between 0 and 1 with steps of 0.05. We measure the entropy of the distribution of opinions. We split of the space of opinions in S = 100 equal squares and calculate the relative frequency $F(o_S)$ of the agents with opinions within each square.

$E = - \sum_{s=1}^{S} F(o_S) ln F(o_S)$

A higher entropy means that the opinions are less concentrated in one area. We see that for low values of acceptance opinions are spread widely, especially if combined with a high rejection threshold. If the acceptance threshold increases, agents start to aggregate in concentrated areas.

If t=1 and u = 1 we see that all the opinions converge to 0.5 since all interactions lead to agents adjusting their opinions towards each other. However, if t = 0.3 and u = 0.2 the opinions start diverging into two groups ending up in a bimodal distribution of opinions being 0 or 1. The reason is that agents come closer if they are not more different than 0.2, while diverging if the difference is 0.3 or more. Using the entropy measure we can see in Figure 1 how the acceptance and rejection threshold values lead to different distributions of opinions. If we have a low acceptance threshold (opinions moving towards each other) and a high rejection threshold (opinions moving away from each other) we will get more diversity of opinions.

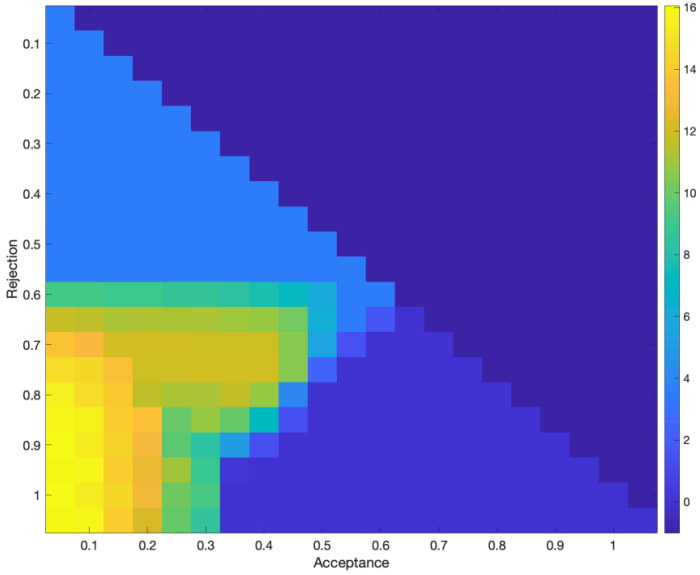

Figure 1: Overview of the entropy at the end of simulation experiments for u varying between 0.05 and 1.0 and t varying between 0.05 and 1.0 (with the constraint t > u) for 30 replications of each tested couple (u, t).

The default model assumes the same probability of interacting with any two agents each time step. What would be the consequence is there is some social structure. For example, agents are located in a physical space and have more likelihood to interact with those who are physically close. Or agents are in a virtual space and are more likely to interact with those who have similar opinions.

We included in the NetLogo model the option to explore different ways agents may interact. You can run a version of the model whether the interactions are based on distance or similarity of opinions. Explore the model and see whether the model leads to different outcomes.

Party competition

We present a simple model of competition among parties hunting for voters are presented by Laver (2005). The model can be downloaded here. Given are n voters and m parties. Voters have a fixed position in a 2-dimensional opinion space and vote for the party that has an expressed position closest (using Euclidean distance). Parties can have one of 3 different strategies:

- Hunter: Did the previous adjustment of the party position increase number of voters? If so, repeat the move. If not, turn 180 degrees around and move forward in that direction
- Predator: If you are the largest party, do not change your position. If you are not the largest party, move your position closer towards the largest party.
- Aggregator: Position is the mean position of voters voting for the party last tick.

The positions of the voters are drawn from a normal distribution, as Figure 2 demonstrates how the parties, the squares, try to position themselves to get a large share of the voters (yellow dots) using one of the three different strategies. Figure 3 shows how the shares of the parties change over time.

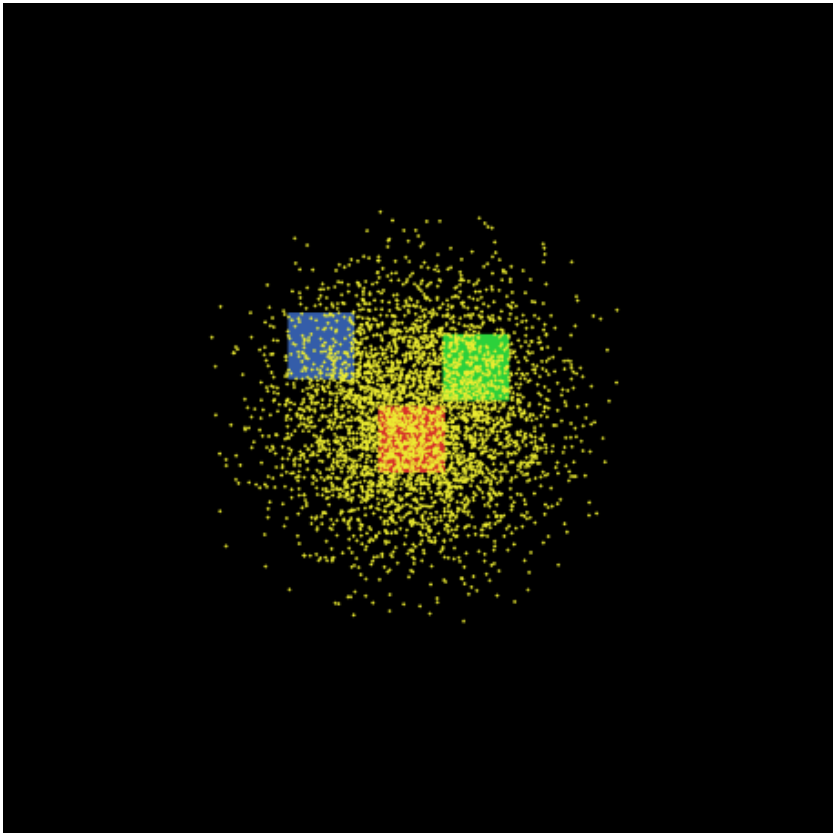

Figure 2: Representation of the competition between parties (squares) to get voters (yellow dots).

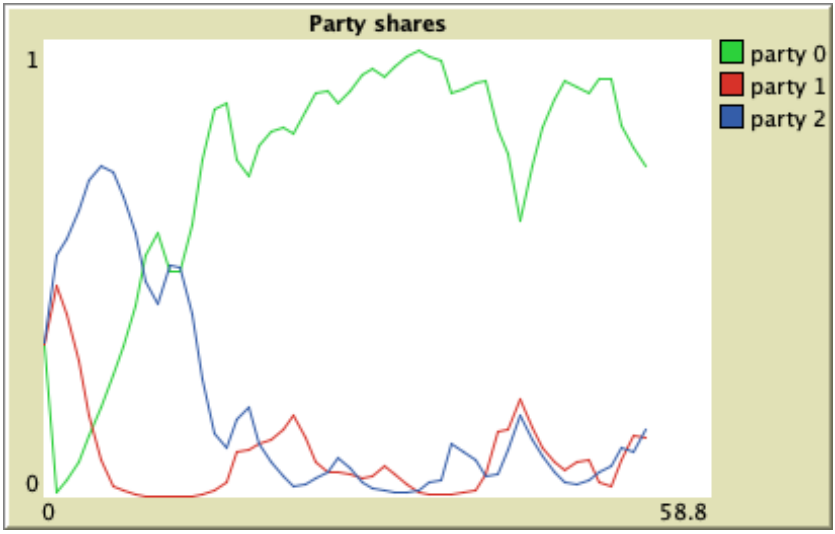

Figure 3: The share of the voters voting for party 0, 1 or 2 each tick, representing an election cycle, or polling event.

Figure 3 demonstrates that there is quite a volatility of the party shares over time. We can measure volatility by calculating the standard deviation of party shares. When we run the model for different configurations of party strategies, we find that a combination of volatility (Figure 4). Three predatory parties lead to the highest level of volatility, which is not surprising since they actively try to become the biggest party. On the other hand, three aggregator parties lead to much more stability.

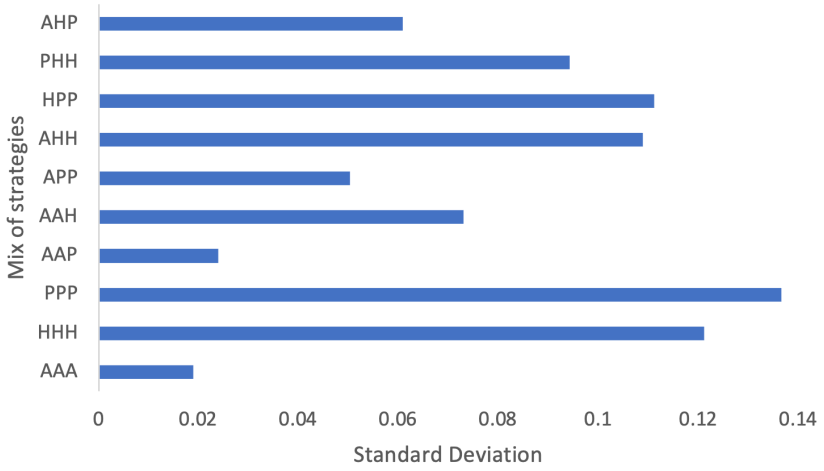

Figure 4: Standard deviation of party shares from 100 runs of 100 election cycles for different configurations of party strategies.

Since the parties are adapting to get a high share of the voters, the average share is similar between the strategies. When we look at different mixtures of strategies, we find that the predatory parties lead to a somewhat bigger party share than the hunter strategy, followed by the aggregator (Figure 5).

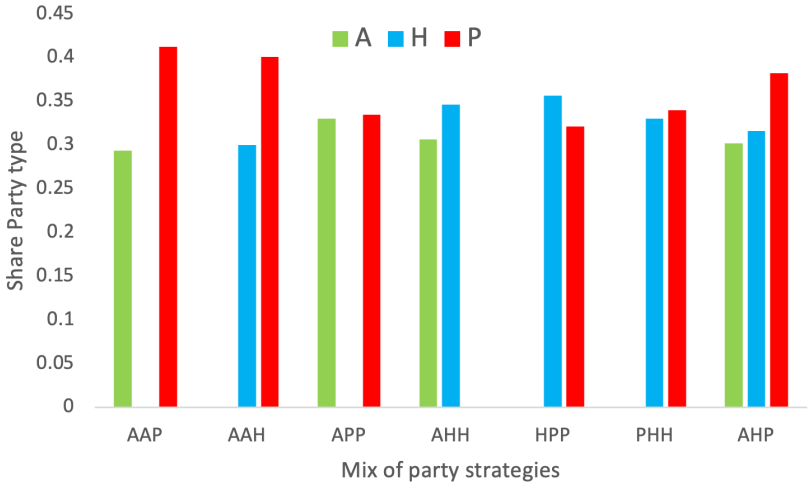

Figure 5: Average party share per party strategy for different configurations of party strategies.

Party competition with opinion dynamics

What might happen in the political competition if agents have no fixed opinions, but change their positions? Here we explore the option to combine the model of opinion dynamics and party competition. The model code can be found here.

The opinion of the voter has two dimensions and we assume that agents move their opinions towards each other if the difference between opinions for both dimensions is smaller than u. However, if the difference in opinions in at least one of the dimensions is bigger than t, the agents start to repel their opinions.

Do positions of parties impact opinion dynamics? We make a simple assumption that if a voter has a nearby party whose position is less than u different from the opinion of the voter, the voter will not change its position. Hence if voters have found a party that fits their position, they will not change their opinion.

Figure 6 shows an illustrative run for a configuration of 3 different types of party strategies when the opinions of voters are fixed.

The parties take turns being the largest party although the hunter strategy is most successful in this example. When we allow for opinions to change, using t=0.6 and u=0.2, we find that one party starts to dominate. Agents having similar opinions as the party position do not change, and parties end up covering different groups of voters in different clusters of opinions.

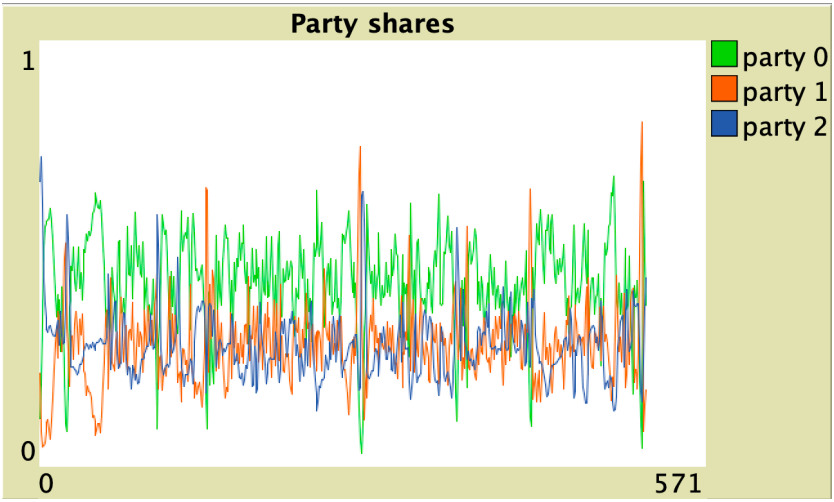

Figure 6: The part shares for an illustrative run where opinions do not change. Party 0 has a hunter strategy, Party 1 has a predator strategy and Party 2 has an aggregator strategy.

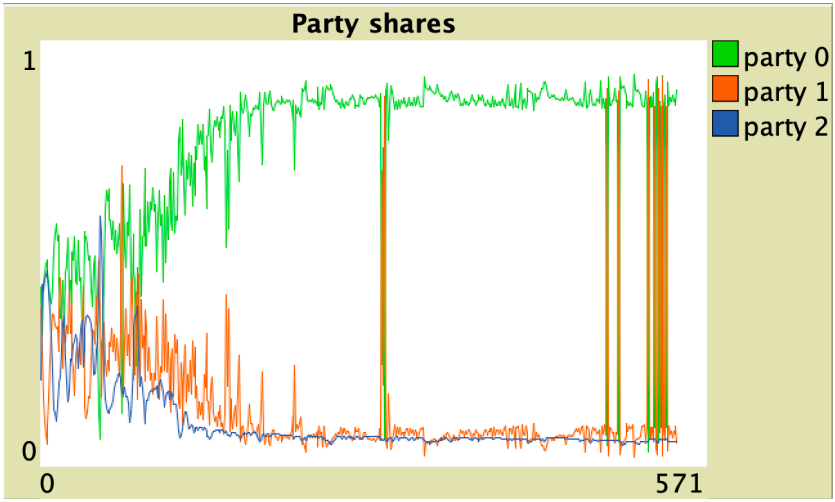

Figure 7: The part shares for an illustrative run where opinions do not change. Party 0 has hunter strategy, Party 1 has a predator strategy and Party 2 has an aggregator strategy.

When we run different mixes of party strategies, we get that for different threshold values of acceptance and rejection, leading to different opinion dynamics, different party strategies will dominate. If agents evolve towards the same opinion the aggregator strategy is the strategy of parties leading to the biggest success (Figure 8). However, if agents' opinions will split into different clusters, predatory strategies will be the most successful (Figure 9). In sum, which type of adaptive behavior of a party is successful also depends on how agents adjust their opinions.

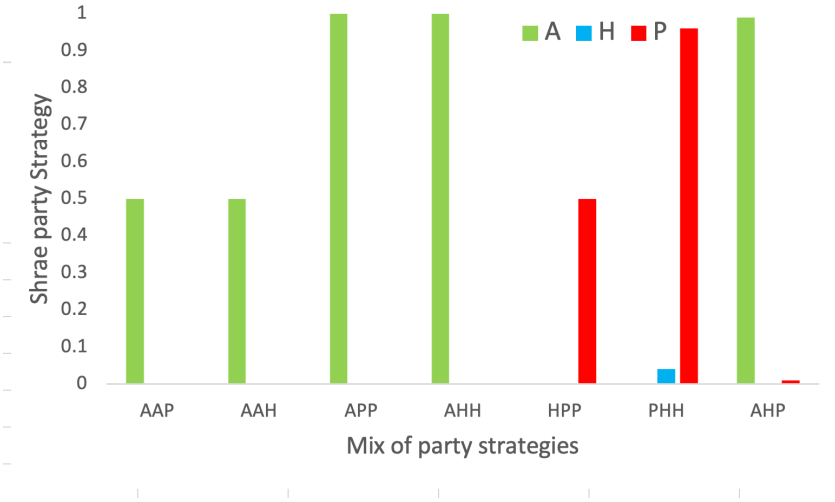

Figure 8: Average party share per party strategy for different configurations of party strategies when u=t=0.6.

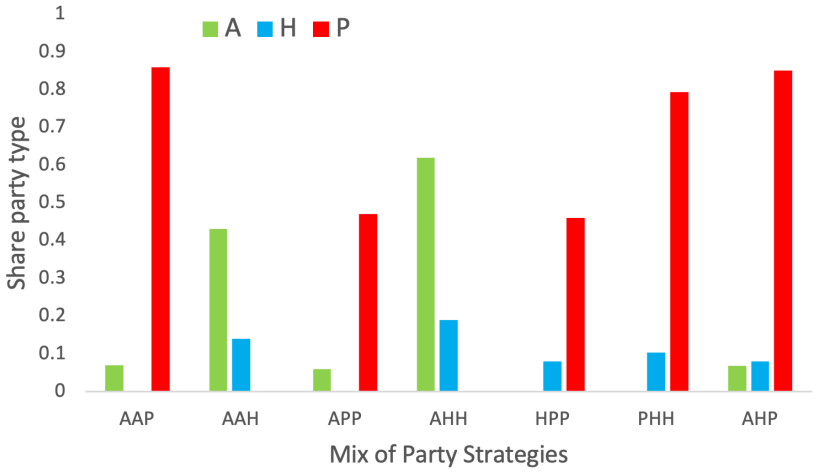

Figure 9: Average party share per party strategy for different configurations of party strategies when u=0.2 and t=0.6.

There is substantial room for extensions of the model. At the moment each tick is an election cycle, while there might be a sequence of polling of a sample of the population and adjustments of positions, as done in Kollman et al. (1992).

Parties might also not be eager to change their position frequently since they have party principles and want to avoid flip-flopping. They may only change if they expect to get a minimum gain of x% of the voters based on the polling data.

We now assumed a central government, but what about local governance. In Kollman et al. (1992) they explored the consequences of voting with your feet. Agents move to other locations if they are unhappy in the current location. This leads to the segregation of opinions in different locations. What if this reinforces opinion dynamics now agents more frequently communicate with those who have similar opinions?

How opinions and politics shape reality

How do opinions impact physical reality? The COVID-19 pandemic demonstrates that opinions about reality impact the actions of people (whether to wear masks or get vaccinated) and therefore change the course of the pandemic. If all citizens followed the recommendations of the governmental health experts, the pandemic would have experienced a different trajectory. Given the uncertainty with many sustainability challenges, we can expect this to happen in the coming decades with many of the big challenges we will experience.

Janssen and de Vries, et al. (1998) developed a model where agents have perspectives on climate change impacting their actions. Agents may change their perspectives on the climate change problem if they are confronted with new information. They may ignore some deviation from their expectations but if their mental models are not explaining the observations, the mental models of the populations change to something that better explains the observations. The model naïvely assumed that policymakers form opinions and make decisions based on facts. We

could update this model with new insights on opinion dynamics and political competition.

There are a number of non-trivial issues to consider. Will agents interact with a fact the same way as an opinion? Even if agents agree on the seriousness of climate change, that does not mean that agents will change behavior? Agents have to consider different issues, and climate change is just one of them. The priority of a topic may change over time. We know from historical studies that institutional changes typically happen after a major event. Thus, agents may find a topic more important if it experiences a negative consequence of not acting. How will this impact something like climate change which is non-reversible?

Summary

In this chapter we showed how opinion and opinion change due to social influence can be modeled, as well as voting on political parties. Moreover, we modeled the adaptation of parties, and the co-evolution of party positions and opinions of voters. Such models might be starting points for including opinions and politics in models of social-ecological systems.

References

Del Vicario, Michela, Alessandro Bessi, Fabriana Zollo, Fabio Petroni, Antonio Scala, Guido Caldarelli, H. Eugene Stanley and Walter G. Quattrociocchi (2016) The spreading of misinformation online. Proceedings of the National Academy of Sciences of the United States of America, 113(3), 554–559.

Hegselmann, Rainer and Ulrich Krause (2002) Opinion dynamics and bounded confidence: models, analysis and simulation, Journal of Artificial Societies and Social Simulation 5(3): https://www.jasss.org/5/3/2.html

Iyengar, Shanto, Yphtach Lelkes, Matthew Levendusky, Neil Malhotra, Sean J. Westwood (2019) The origins and consequences of affective polarization in the United States. Annual Review of Political Sciences 22: 129–146.

Janssen, Marco and Bert de Vries (1998) The Battle of

Perspectives: a multi-agent model with adaptive responses to climate change, Ecological Economics 26(1): 43-65.

Kollman, Ken, John H. Miller, and Scott E. Page (1992) Adaptive Parties in Spatial Elections. The American Political Science Review 86(4): 929–37.

Laver, Michael (2005) Policy and the Dynamics of Political Competition. American Political Science Review 99(2):263–81.

Noorazar, Hossein (2020) Recent advances in opinion propagation dynamics: a 2020 survey, The European Physical Journal Plus, 135: 521: https://doi.org/10.1140/epjp/s13360-020-00541-2

PART VI

CLOSING

CHAPTER 22

Closing

You know come to the end of the current version of the book. The book will be updated on a regular basis with new and updated chapters. Check out the book website https://intro2abm.com/ for latest updates and send me any comments or suggestions via Marco.Janssen@asu.edu.