NORTHWESTERN UNIVERSITY

Genetic Algorithms for the Exploration of Parameter Spaces in Agent-Based Models

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Forrest J. Stonedahl

EVANSTON, ILLINOIS

December 2011

ABSTRACT

Genetic Algorithms for the Exploration of Parameter Spaces in Agent-Based Models

Forrest J. Stonedahl

This work provides the first comprehensive investigation of the use of genetic algorithms for exploring the range of behaviors produced by agent-based models. Agent-based modeling (ABM) is a powerful computer simulation technique in which many agents interact according to simple rules resulting in the emergence of complex aggregate-level behavior. However, as ABM is increasingly employed in both natural and social sciences, the methods and tools for understanding, exploring, and analyzing the behavior of agent-based models have not kept pace. In particular, models may be characterized by a large number of parameters, and the task of discovering parameter settings for which a model will produce a certain behavior is both difficult and time-consuming. Genetic algorithms (GAs) offer a flexible metaheuristic search mechanism which has previously been successful in a variety of combinatorial optimization and search problems. There is a rich space of possible model exploration tasks, and we offer a new unified framework for the creation and application of quantitative measures to perform these tasks using an evolutionary-search paradigm. We demonstrate the utility of GAs for ABM parameter exploration through a sequence of case studies in various application domains, including behavioral biology, viral marketing, archeology, and web-based journalism. This work advances agent-based modeling methodology by exploring when and

how GAs can be useful in the model development and analysis process. It also contributes to a deeper understanding of GAs, by evaluating their strengths and weaknesses with regard to the particular challenges posed by this problem domain. We develop novel heuristics for dealing with model stochasticity in conjunction with fitness caching techniques. We also present the first set of benchmark models/tasks for automated ABM parameter search and exploration, and we rigorously investigate the performance of GAs on these benchmarks, with varying levels of stochasticity. An important product of this research is BehaviorSearch, a new automated software tool for efficient exploration of ABM parameter spaces. The design and affordances of BehaviorSearch are discussed with respect to improving model exploration and analysis by ABM practitioners.

Acknowledgments

First and foremost, I thank my wonderful wife Susa, who insisted that I not get all mushy in these acknowledgements. So I won't. There is insufficient space to give her the thanks she deserves in any case. Second and still foremost, I thank our Siamese cat, Gabby, who insisted that I thank her in these acknowledgements. Either that or she was just meowing because I forgot to feed her in my thesis-writing haze. Sometimes it's hard to know.

It goes without saying that this thesis could never have come to fruition without the support of my advisor, Uri Wilensky, who first introduced me to the marvelous world of agent-based modeling, complex systems, and emergence. Uri also possesses an uncanny ability to gather together a research group composed of some of the most intelligent, quirky, and amazing people I have ever met. I am deeply honored to have been a part of the Center for Connected Learning and Computer-Based Modeling these past years. It has been a place of much intellectual growth, and I will never forget it. I thank everyone in the CCL – for the thought-provoking conversations, the good advice, the frenetic Google-document-sharing chats, the late-night 3D-printer surgeries, the inspiration to improve education and technology, and all of the laughter along the way. I particularly wish to single out Michelle Wilkerson-Jerde, who walked side by side with me this past year on the simultaneously treacherous paths of thesis writing and academic job searching. Special thanks also goes to Josh Unterman and Daniel Kornhauser who each, in their own inimitable ways, shared oysters of wisdom with me.

I am also deeply grateful to the other members of my committee: Doug Downey, who always gave me sensible advice whenever I asked[1], Luis Amaral, who taught me about networks (and whose book recommendations were superlative[2]), and Bill Rand with whom I have had the pleasure of collaborating on numerous evolutionary computation projects, and who has served as a mentor to me on countless occasions.

Thanks to my father for writing his daily blog, and helping keep this *Forrest* in touch with his Idaho *roots*. Thanks also to my mother for unswervingly positive support, and proofreading portions of this thesis, as well as several of my other papers along the way. (Who besides a loving mother could enjoy reading a stream of text filled with largely unintelligible jargon?) I also thank my brother Birrion, whose ski bum philosophy on life helps to counterbalance my workaholic nature, and thus I'm sure maintains some crucial harmony of the universe. I would also like to mention Marion, Joyce and Jack, Karline, Susan, Peggy, and all those among my extended family and in-laws who have been supportive during my graduate school years.

I thank my friend Dave Ohls, for the many empathetic and commiserative chats about the ups and downs (perhaps mostly the downs) of graduate school. And David and Laura Little for their noble attempts to preserve my sanity with the occasional enjoyable evening of card/board games.

Thanks also to Valdis Krebs, for his stimulating email discussions and his generosity in sharing an interesting dataset with me. I thank Janet Pierrehumbert and Robert Daland, for their collaboration on the linguistics modeling project that is briefly mentioned in this text. I thank Bryan Pardo for wandering into my office with occasional advice on teaching,

[1] proving that I should have asked more often...
[2] see, e.g., *Influence: Science and Practice* by Robert Cialdani

research, job hunting, and random trivia. I am very grateful to NICO (the Northwestern Institute on Complex Systems) for fostering interdisciplinary research and collaboration at Northwestern.

I also must thank the miracles of modern technology – this thesis work would never have been possible without many many CPU-hours churning through millions of agent-based simulations. For the availability of these computational resources I especially want to recognize: Luis Amaral, who generously provided time on his computing cluster, Northwestern's Social Sciences Computing Cluster, the University of Maryland's OIT HPCC, and Northwestern's new *Quest* high performance computing cluster (which executed the bulk of the experiments reported herein). I am also thankful to several sources of funding, which supported this work at various times and in various ways: specifically, the National Science Foundation (grant IIS-0713619), a Murphy Society grant from the McCormick School of Engineering, and a Google and WPP Marketing Research Award. Also, I am grateful for the generosity of both The Graduate School and the Cognitive Science Program at Northwestern, in providing supplementary travel grant awards that allowed me to present this work at several conferences and workshops, and thus receive constructive feedback from my academic peers at other institutions.

Finally, I thank everyone who I meant to thank but have forgotten in these acknowledgements (which is now, by virtue of self-reference, the empty set).

Preface

"... from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved."

– Charles Darwin

Though I can't remember precise words and must plead for some artistic license as I paraphrase them below, I can still vividly recall one afternoon several years ago in my thesis advisor's office, having just expressed a bit of the traditional graduate student angst about my chosen thesis topic. "Well," my advisor responded, "it all depends on whether you want to have a *tidy thesis* or a *messy thesis*. (I must have looked aghast – after all, who in their right mind would want to produce a *messy thesis*?) "Don't get me wrong", he continued, "my personal preference is toward messy theses. Tidy theses are narrowly-defined, for example: 'we present a novel algorithm that performs X% better than all previous algorithms for a specific problem Y'. Messy theses are much broader in scope, expressing powerful ideas and looking for the *big picture*."

Both types of theses are valuable. In the field of artificial intelligence, we often discuss problem solving strategies in terms of *exploration* versus *exploitation*. For instance, if you are going out to dinner, should you order a slight variation of your favorite dish (*exploitation* of previous knowledge with high likelihood of modest reward), or try a new and exotic dish (*exploration* in the hope of discovering something far better, but the reward is uncertain).

Is it better to take a well-known travel route and refine it, or go out hunting for that elusive 'Northwest Passage', which might or might not exist? All good theses contribute something new to their discipline, and the research process always contains some mix of exploration and exploitation. But in my view, given this spectrum, "tidy theses" put more emphasis on the exploitation side, while "messy theses" expend more effort exploring. Perhaps as a result of my name, I also have a penchant for arboreal analogies. "Tidy theses" are akin to pine trees, thin spikes reaching straight up toward the sky. "Messy theses" resemble oak trees, spreading out as they reach upward. Or possibly mangroves, which also reach down, around, and every which way with their roots. But since this thesis is about evolution-inspired algorithms, perhaps a better metaphor would be the phylogenetic *tree of life*. This tree embodies a grand exploration of divergent speciation, fraught with the extinction of unproductive branches, graced by the explosion of new life forms arising in unexpected areas, and sometimes captivating by, as Darwin put it, the "endless forms most beautiful" that have arisen from so simple a beginning. It is my hope that the intrepid reader of this "messy thesis" will see beyond the occasional knotted root or twisted branch, and emerge from the experience with a clearer sense of the whole tree, as well as a few thoughtful seeds that may bear fruit in scientific explorations of the future.

$\sim$ *Forrest Stonedahl*

Originality of materials

In the preparation of this dissertation, I drew upon several of my previous papers. I would be remiss if I did not give proper credit to my co-authors, to whom I am greatly indebted for their collaboration on these projects which contributed to my thesis work. Specifically:

- Chapter 4 is adapted from: Stonedahl, F. & Wilensky, U. (2011). Finding Forms of Flocking: Evolutionary Search in ABM Parameter-Spaces. *Multi-Agent-Based Simulation 2010*, T. Bosse, A. Geller, & C. M. Jonker (Eds). Lecture Notes in Artificial Intelligence 6532. pp. 61–75. Springer, Heidelberg.
- Chapter 5 is adapted from: Stonedahl, F., Rand, W., & Wilensky, U. (2010). Evolving Viral Marketing Strategies. *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO '10)*. July 7-11. Portland, OR.
- Chapter 6 is adapted from: Stonedahl, F. & Wilensky, U. (2010). Evolutionary Robustness Checking in the Artificial Anasazi Model. In *Proceedings of the AAAI Fall Symposium on Complex Adaptive Systems: Resilience, Robustness, and Evolvability*. November 11-13, 2010. Arlington, VA.
- Chapter 7 is adapted from: Stonedahl, F., Anderson, D., & Rand, W. When Does Simulated Data Match Real Data?: Exploring Model Calibration Functions using Evolutionary Computation. *Poster presented at GECCO '11*. July 12-16. Dublin, Ireland.
- Chapter 8 is adapted from: Stonedahl, F. & Stonedahl, S. H. (2010). Heuristics for Sampling Repetitions in Noisy Landscapes with Fitness Caching. *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO '10)*. July 7-11. Portland, OR.

All other material presented herein is original.

Dedication

In loving dedication to my paternal grandparents,

Marion and Jake Hvistendahl,

who encouraged, inspired, and financially supported me on the path to higher education...

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction and Motivation

"It [the computer] is the first metamedium, and as such it has degrees of freedom for representation and expression never before encountered and as yet barely investigated."

– Alan Kay

"Revolutions always come around again. That's why they're called revolutions."

– Terry Pratchett, *Night Watch*

1.1. Backdrop

We are in the midst of a scientific revolution, resulting from the integration of efficient, ubiquitous, and inexpensive computation into everyday scientific practice. Perhaps in recent years the word "revolution" has been so well-worn that it has lost much of its meaning. Our society is full of pundits and aspiring visionaries declaring how the world, the nation, the internet, or your routine kitchen cleaning is being (or can be) revolutionized by new products or technologies. Nonetheless, I contend that this is a true revolution, and that this revolution has important consequences for the future. Admittedly, digital electronic computers have been employed in science and engineering settings since their invention in the early 20th century (and earlier computing machines, such as mechanical calculators or Babbage's programmable "difference engine" which go back centuries further). However, for many years, in the majority of scientific practice, computers were used as little more than

glorified calculators, capable of evaluating far more complicated mathematical expressions than were previously possible, and of crunching data more accurately and efficiently (by many orders of magnitude) than the human "computers" they had supplanted. Although this capability alone was a giant breakthrough that permitted scientific endeavors to be undertaken on a previously unimaginable scale, it does not represent the fundamentally qualitative change which computers are currently making on the way science is conducted. That qualitative change stems from computer's universal flexibility – the potential for it to act like *anything else*. This is by no means a new idea: pioneers of computing like Alan Turing and John von Neumann fully recognized this incredible power, and Alan Kay and Seymour Papert stand out as more recent advocates of the protean nature of computers. This nature is what allows computers to "simulate" other systems, both real and imaginary, and it permits the creation of computational models of natural and social phenomena.

However, it is only in the past couple of decades that this form of simulation-based computer modeling, rather than purely mathematical modeling, has really taken hold and invaded everyday scientific practice. Whereas 20-30 years ago, science experiments were being performed *in vivo*, *ex vivo*, *in vitro*, and *in situ*, today more and more experiments are being performed *in silico*. This is no longer merely running statistical regressions and finding trend lines, but more explicit simulations from first principles, from axioms, from underlying assumptions, and from observed interactions. And perhaps most striking, it is invading social science fields traditionally considered to be less quantitative such as anthropology, archeology, sociology, psychology, political science, and others. It would be simpler to enumerate the scientific fields that are unaffected by this paradigm shift in computational modeling. I believe there are none. This is not to say that simulation is the best approach to solve any question in any field – only that every field has its share of questions that can be

approached through simulation. The ability to create virtual models of any phenomena being studied is incredibly useful, and although this has been known to a few for a long time, it is being continually rediscovered by scientists and researchers across the globe. The computer's capacity to be transformed into a microworld that mimics aspects of reality, based on the rules that we prescribe for it, permits not just *more science*, but *different science*. Wolfram [2002] argued along these lines in his influential book *A New Kind of Science*, although he latched specifically onto cellular automata, which is only one type of constructive computational modeling, whereas I would argue that cellular automata are often too constraining to use to model most real world systems, and that other approaches, such as agent-based modeling (discussed below), usually provide a more flexible approach. Some have made an analogy between the current transition from traditional equation-based models to algorithmic computational models and the transition from the Roman numerals to the Arabic numerals for representing numbers; such transformative shifts require a "restructuration" of existing knowledge representations, and the result is new paradigm for thinking about and undertaking science [Wilensky & Papert, 2010; Wilensky, 2006]. The broader point is that the possibilities for virtual experiments go far beyond those constrained by real-world experimentation, and they can often provide more explanatory power than mathematical equations and statistical correlations. There have been many factors contributing to the current rise of constructive simulation-based computing in science, but several stand out.

(1) The availability of inexpensive and increasingly powerful computing has fueled the increased adoption.

(2) The creation of supporting tools and languages (e.g., [Wilensky, 1999, 2000; Resnick, 1994; Resnick & Wilensky, 1993; Wilensky & Resnick, 1999]) have lowered the threshold for developing such models, especially for non-expert programmers.

(3) Rather ironically, despite academia's general tendency toward politically liberal viewpoints, it can be staunchly conservative within its own domain. Even today, many fields continue to reject new methodological tools such as computational models, clinging fiercely to long-established equation-based techniques. Slowly (but surely, I believe) the tides are turning, as the new methods diffuse through the social network that comprises the scientific community.

This setting provides the backdrop for the work of this dissertation, which will focus only on one specific form of computational modeling (agent-based modeling), and will investigate the affordances of one specific method (genetic algorithms) of analyzing and exploring the behavior of such models.

1.2. Agent-Based Modeling

Agent-based modeling[1] (ABM) is a powerful simulation technique that is being increasingly applied in a wide range of scientific research endeavors [S. Bankes, 2002; Berry, 2002; Bryson, Ando, & Lehmann, 2007; Harrison, Lin, Carroll, & Carley, 2007; N. Gilbert & Troitzsch, 2005; Huang, Xiang, Madey, & Cabaniss, 2005; Wilensky & Rand, in press]. Simultaneously, agent-based modeling is emerging as its own area of research, focusing on the study of ABM methodology and associated techniques, rather than on its application to any particular field. In agent-based models, many agents interact according to simple rules,

[1] sometimes alternatively called individual-based modeling, or multi-agent simulation

resulting in the emergence of complex aggregate-level behavior. The emergence of qualitatively different behavior at the aggregate-level than at the level of its constituent parts is a hallmark of complex systems, which has been discussed both in the academic sphere [P. Anderson, 1972; Simon, 1973; Bar-Yam, 1997] and introduced to lay audiences through a number of popular books [Waldrop, 1992; Gell-Mann, 1995; J. H. Holland, 1995; Mitchell, 2009]. Attempts to formally prove theoretical properties of these complex agent-based systems rarely succeed, and in most cases more empirical methods of analysis and testing are necessary [Edmonds & Bryson, 2004]. However, the number of controlling parameters (and range of possible values) for agent-based models is often large, the computation required to run a model is usually significant, and the models are predominantly stochastic in nature, meaning that multiple trials must be performed in order to assess the model's behavior (see Figure 1.1, for example). These factors combine to make a brute-force exploration (or "factorial design" experiment) of model behavior infeasible. Furthermore, agent-based models may constitute complex systems in which the interactions between parameter settings are highly non-linear, so testing the effects of varying individual parameters separately is not a reliable approach for predicting the effects when multiple parameters are varied simultaneously. Despite those caveats, the simplicity of these analytic approaches (factorial and univariate) has resulted in widespread use throughout the modeling community.

In my estimation, the current state of affairs is grim indeed. At a recent conference I attended, I was shocked when one of the keynote speakers (who shall remain nameless), admitted that he usually did not perform any type of parameter variation (not even basic factorial sweeps) even though he is in the business of using agent-based models to advise the U.S. Department of Defense on a variety of high profile issues. Unfortunately, this anecdote highlights that the problem is in part a cultural one. Regardless of what new and improved

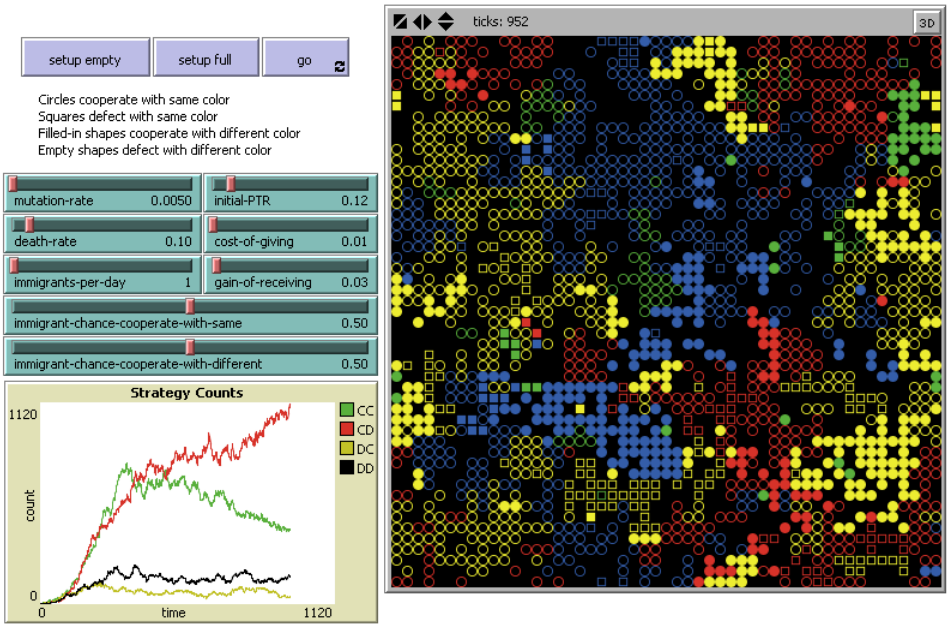


Figure 1.1. The NetLogo Ethnocentrism model [Wilensky & Rand, 2003; Axelrod & Hammond, 2003]. In this case, there are eight model parameters, each of which may take on a large range of values. One parameter, *immigrants-per-day*, is integer-valued, while the other seven are real-valued (continuous). If each parameter may vary between 10 possible levels, the size of the search space would be 10^8, and evaluating a single point in that space requires multiple simulation runs (replicates) since the model is not deterministic. Assuming that it requires only 10 replicates, and each run takes only 10 seconds, a complete exploration of this search space would take 10^{10} seconds, or approximately 317 years for a single processor.

model exploration techniques we propose, or how user-friendly we can design the software for performing these tasks to be, issues like these will persist until the cultural norms of the agent-based modeling community have shifted to give appropriate weight to the analysis of model parameters as part of the verification and validation within the modeling process [Wilensky & Rand, 2007, in press]. At present, the sophistication of parameter exploration methods varies widely across and within modeling communities, and in most cases best practices have not been established. However, the socio-cultural battle for increased rigor

in agent-based model analysis is outside the scope of this thesis, and I am optimistic that the availability of more advanced (yet easy-to-use) methods of exploration (such as those promoted in this document) will break down some of the technical barriers to change, and help to move this scientific community in the right direction.

While simple methods like factorial sweeps are adequate in some cases, there is a large class of agent-based models for which the parameter space is too large and the interactions between parameters are too complex. Thus, more sophisticated search techniques are needed to explore and discover interesting areas of the parameter space. Genetic algorithms [J. Holland, 1975; D. E. Goldberg, 1989] are one natural choice, since they have proven to be successful in numerous nonlinear combinatorial search/optimization problems. The general idea of genetic algorithms will be introduced in Section 1.4 below, as well as explained in greater detail in the specific context in Chapter 3.

1.3. Illustrative Example

In order to discuss the general problem of model exploration in a more concrete manner, let us elaborate one example that demonstrates the utility of searching the parameter-space of an agent-based model. In particular, consider an agent-based model of airplane boarding, similar to the simulations performed by Ferrari and Nagel [2005], or Capelo et al. [2008]. In such a simulation, each passenger is modeled as an agent with varying characteristics (assigned seat, number of carry-on items), who will enter the aircraft (according to various boarding-group schedules), move toward their seat, place their luggage in an overhead bin, and sit down. As any frequent flier knows all too well, this process may be delayed by standing in the aisle, waiting for other passengers to get out of the way, getting up out of a seat to let someone else pass to their window seat, etc. One goal of creating a simulation could be

to determine what policies an airline company can make that would hasten and/or smooth the boarding process (as in [Ferrari & Nagel, 2005]). However, social scientists might wish to study the process of airline boarding for entirely different reasons. Psychologists might wish to examine frustration levels as a result of delays in the distributed queuing process. Sociologists or anthropologists might be interested in using airplane boarding studying cultural differences in pairwise spatial distances between strangers in a constrained environment. Economists might want to investigate price discrimination in seating order, and the differential utility in going earlier rather than later in the boarding process. In each of these cases, an agent-based model of airplane boarding would allow scientists to explore counter-factual scenarios, test the sufficiency of various hypotheses as generative explanations for the observed behavior, etc. For an extended (and much more eloquent) discussion of the reasons for and benefits of creating agent-based models (which go significantly beyond prediction), I point the reader to the excellent short paper "Why Model?" by Joshua Epstein [2008].

Regardless of the purpose of model creation, there are a number of parameters associated with an ABM such as this. First, there are parameters that the airline may have some control over (boarding seat schedule, number of carry-on items allowed, amount of intervention by airplane stewards in the boarding process, floor plan of aircraft, etc). It might be useful to find choices for these parameters that yield minimal time for the boarding process. This is a classic case of "simulation optimization", but in fact model exploration extends beyond optimization in many ways, as will be discussed further in Chapter 3. One might also be interested in finding so-called *leverage points* [Forrester & Collins, 1969; Meadows, 1997; J. Holland, 2008] in the space, where a small increase in one (or several) parameters can yield a considerable change in system behavior (e.g., a drastic decrease in boarding time). Second, the model may also have numerous parameters that the airline has little or no influence over

(passenger movement rate, passenger's preferred interpersonal distance, number of children on the flight, number of passengers that arrive late at the gate, time required to step aside for another passenger by to get to their center or window seat, time required to place bags in an overhead bin as a function of unused bin capacity, etc). While some of these parameters can be estimated by field observations and historical data, there will inevitably be uncertainty about some of these values.

When suggesting a policy decision on the basis of simulation results, it is important to know how poorly that decision might fare as a result of incorrect estimates of these parameters. To achieve this, we might perform a worst-case multivariate sensitivity analysis by searching for parameter settings (within the parameters' feasible ranges) that yield the slowest aircraft boarding scenarios. Another relevant task is model parameter calibration or "tuning". In particular, if a data set is available that provides information on how long boarding times have historically taken, then it would be possible to search for parameter settings that closely match the real-world data by attempting to minimize the error between simulated and real results. This act of calibrating the model with empirical data can assist in the process of model validation. Additionally, search can be used to test the model for conceptual flaws or programmatic errors. For instance, when searching the parameters for minimal boarding time scenarios, if the search method discovers parameter settings such that a Boeing 747 can board all passengers in less than two minutes, either the simulation model is faulty (possibly due to a programming bug), or one (or more) of the parameters being considered is falling seriously outside the ranges that are feasible in the real world. This line of thought opens up avenues for automated model testing via search. In short, searching a

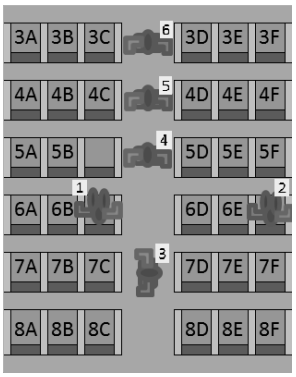


Figure 1.2. This is an example visualization of an agent-based model of aircraft boarding, which is a reproduction of *Figure 2* from Capelo et al. [2008]. In this simulation each passenger is modeled as an individual agent that can interact with other agents. This is in contrast to aggregate-based modeling techniques, such as using differential equations to describe the rate passenger flow into the cabin as a function of the number of passengers already seated.

model's parameter space for extreme results can lead to the discovery of interesting information about model behavior that can be useful in various phases of model development, testing, and analysis.

Although I have only discussed a single example from the airline industry, agent-based modeling is applicable to a wide variety of fields. A brief (and necessarily far from exhaustive) alphabetic sampling from the literature includes examples of ABM in: anthropology [Axelrod & Hammond, 2003], archaeology [Axtell et al., 2002], bioterrorism [Carley et al., 2006], business management [North & Macal, 2007], ecology [Grimm & Railsback, 2005], education [Abrahamson, Blikstein, & Wilensky, 2007], marketing [Rand & Rust, 2011], materials science [M. Anderson, Srolovitz, Grest, & Sahni, 1984], medical research [An & Wilensky, 2009], military tactics [Ilachinski, 2000], neuroscience [Wang et al., 2008], political science [Epstein, 2002], social policy [Maroulis et al., 2010], sociology [Schelling, 1969], urban development and land use [Brown, Page, Riolo, Zellner, & Rand, 2005], and zoology [Bryson et

al., 2007]. In addition, agent-based modeling appears to be playing an increasing role in the burgeoning cross-disciplinary field of "network science" (see, e.g., [Guimera, Uzzi, Spiro, & Amaral, 2005; Holme & Newman, 2006]). In some of these fields, the models that are created may emphasize exploring controllable model parameters for the sake of discovering policies and making decisions, while in others the focus may tend toward model calibration, testing, and verisimilitude with respect to real-world data. However, regardless of the discipline in which the agent-based modeling is practiced, some form of automated parameter exploration can benefit the process of understanding or analyzing these models.

1.4. Genetic Algorithms

Genetic algorithms [J. Holland, 1975] belong to a family of evolution-inspired algorithms, such as evolutionary strategies [Rechenberg, 1973] and evolutionary programming [L. J. Fogel, 1966], that have been invented (independently) in Europe and the United States. More recent variations on evolutionary algorithms include genetic programming [Koza, 1992], differential evolution [Storn & Price, 1997], and grammatical evolution [C. Ryan, Collins, & Neill, 1998]. The rise of bioinformatics sometimes leads the uninitiated to believe that genetic algorithms (GAs) involve the application of computer science algorithms to the decoding of the DNA/genome of biological species. However, the situation is quite the opposite. Instead, genetic algorithms result from the application of principles of biological evolution to computer science, to form a domain-independent problem solving technique. In other words, genetic algorithms seek to "evolve" solutions to challenging problems by artificially mimicking the forces of variation and natural selection on a reproducing virtual population of candidate solutions. Because of its generality and domain-independence, genetic algorithms is described as a "meta-heuristic" search algorithm.

Because genetic algorithms have proven successful on a wide range of nonlinear combinatorial search/optimization problems, they form a natural choice for a search mechanism for exploring the behavior of computer simulations (and agent-based models more specifically). In fact, one of the earliest applications of genetic algorithms was in optimizing parameters for a simulation of a living cell [Weinberg, 1970], and there have since been many various applications of GAs to parameter optimization problems (e.g., [Grefenstette, 1986; Bäck & Schwefel, 1993]). Additionally, previous studies by myself [Sondahl & Rand, 2007] and others [Mitchell, Crutchfield, & Das, 1996; Packard, 1988] have demonstrated GA's success when evolving rules for cellular automata, which can be considered a restricted subclass of parameter search in agent-based models. Several different researchers have either tried or suggested the use of genetic or other evolutionary algorithms for ABM parameter search tasks [Caporale, Serguieva, & Wu, 2009; Heppenstall, Evans, & Birkin, 2007; Narzisi, Mysore, & Mishra, 2006; Calvez & Hutzler, 2005], but the bulk of prior work has focused on answering specific questions regarding a single specific agent-based model. Related work in this area will be discussed in much greater detail in Chapter 2; however, in summary, this idea has received scant systematic attention thus far.

1.5. Overview of Document Structure and Contributions

This thesis document is structured as follows. Chapter 2 provides the first comprehensive literature review of research in this area, tying together themes from the exploration and analysis of agent-based models to relevant work in the area of genetic algorithms and metaheuristic search. Chapter 3 proposes a theoretical and methodological framework called "Query-Based Model Exploration" (QBME), which provides the necessary structure for synthesizing the various ideas and concepts that are demonstrated in the case studies that follow.

The development of this comprehensive framework is, in itself, a contribution to the field. Chapters 4-7 provide case studies that illustrate how and why genetic algorithms can be an effective exploratory mechanism for the parameter-space of ABMs. Each of these case studies is essentially written to stand on its own, providing sufficient background in both the case study domain and the relevant ideas of evolutionary ABM exploration. Chapter 4 demonstrates the use of exploratory searches to discover the range of behavior produced by models of collective animal movement (i.e. "flocking" behavior). Chapter 5 shows how genetic algorithms can be used to explore a model of diffusion of innovation in social networks, in order to a) find good strategies for viral marketing campaigns, and b) discover intriguing differences between how real and abstract social network structures interact with this agent-based model. Chapter 6 tackles the issue of model calibration and sensitivity analysis, using genetic algorithms to search the parameter space of the well-known "Artificial Anasazi" simulation. Chapter 7 delves further into the question of how different calibration measures may be more or less effective as fitness functions for driving the evolutionary search process, in the specific context of matching a new agent-based model of consumer behavior in online news browsing with a real-world dataset. Each of these case studies makes a contribution to its specific domain area, as well as illustrating principles and general ideas about the broader topic of using genetic algorithms for exploring agent-based models. Following the case studies, Chapter 8 transitions into a more abstract/mathematical treatment of the problem of stochasticity in agent-based simulation, and how this "noise" affects the effectiveness of search processes that use "fitness caching" to reduce redundant computation. This leads naturally into Chapter 9, which provides a comparison of genetic algorithms performance (vis-à-vis other search algorithms) on a variety of benchmark model analysis tasks, with and without fitness caching, and with varying levels of repeated sampling (for noise reduction).

This chapter's contributions are at least two-fold: 1) the creation of a set of benchmark model exploration tasks provides a valuable baseline for future research in this area, and 2) this work provides the first comprehensive comparative study of genetic algorithms performance for this type of task. It also provides a basis for evaluating the mathematical measures of noise derived in Chapter 8. It is my belief that theoretical research should be grounded in real-world problems, and conversely, that theoretical results should inform and enhance the practice of the field. Accordingly, Chapter 10 discusses the design of a practical software tool (*BehaviorSearch*) that I have developed and released in order to bring the QBME methodology for model exploration and analysis (described in Chapter 3) within the reach of the greater scientific modeling community. *BehaviorSearch* incorporates insights from my theoretical research into the design of a low-threshold tool for the automated exploration of agent-based model behavior, and it interfaces with NetLogo [Wilensky, 1999, 2001; Tisue & Wilensky, 2004], which is a premier agent-based modeling language and integrated modeling environment. Finally, Chapter 11 provides some closing remarks and shares several ideas for future research that I believe will be fruitful, based on the perspective I have gained through this experience.

This thesis centers around one underlying question: do genetic algorithms provide an effective search technique for exploring the parameter spaces of agent-based models (ABMs)? I claim the answer to this question is "yes". However, to fully address this question, a simple "yes" or "no" answer will not suffice; and thus I will dissect it into a series of more specific questions:

- How can genetic algorithms be used to answer the type of questions that modelers are concerned with?

- What type of modeling analyses are amenable to this approach?
- What results have genetic algorithms found on real research modeling tasks?
- How effective are genetic algorithms relative to comparable techniques?
- What factors influence their efficacy, and how might they be improved to better address the particular challenges posed by this problem domain?

Along the way, I will also justify why this research direction is worth pursuing, using a combination of real-world modeling problems and classic/abstract models drawn from the complex systems research community. In particular, I will argue for (and demonstrate) the utility of integrating intelligent search/optimization techniques into the practice of agent-based model exploration and analysis.

CHAPTER 2

Literature Review

"We build too many walls and not enough bridges."
– Isaac Newton

"A person who won't read has no advantage over one who can't read."
– Mark Twain

The question of how to explore and analyze the behavior of agent-based models (as well as other types of computer simulation) is a broad one, and it has been approached from different directions by different researchers. Sometimes there is a tendency in the various academic disciplines to carry out research that is isolated from other disciplines, and publish in their own field's specialized journals and conferences. Whether conscious or not, this tendency can result in creating walls between research carried out in one domain and another. However, since agent-based modeling is an inherently cross-disciplinary methodology[1] I will attempt (through this literature review) to bridge some of the gaps between disparate communities and offer a holistic survey of the relevant ideas in play. I will first discuss some of the methods suggested for exploring models in general, before turning to *search-based* methods in particular. This will be followed by a discussion of research from the genetic algorithms literature that is appropriate to the agent-based model exploration problem domain.

[1]ABM is similar in this regard to fields like statistics, which cut across disciplinary boundaries. However, unlike statistics, which is relatively mature, agent-based modeling is quite young and has (as yet) few dedicated journals and conferences. Perhaps in the future, ABM research will become less fragmented.

2.1. Overview of General Methods for Model Exploration

A first approach to exploring the parameter-space of agent-based models is through human interaction and/or supporting visualization tools. It is common practice for researchers to experiment with different settings of their models, according to their intuitions. By using knowledge and intuitions about the inner workings of the model, humans are often able to more efficiently navigate the parameter-space than computer algorithms which treat the model as a "black box". Fehler et al. [2006; 2004] have suggested a methodology they call "white box" calibration, in which humans use their knowledge of the model's structure to decompose the model into smaller units, thus effectively reducing the parameter-space that must be considered. While "white box" approaches provide clear benefits, in many cases model decomposition may be extremely hard or even impossible. Another possibility is to let humans view and process the results of evaluating points in the search space, form their own mental models of the shape of the space, notice trends or search for points (parameter-settings) of interest. The greatest challenge here is to help humans make sense of the high-dimensional space by providing visualizations that humans can translate into an understanding of model behavior. A full survey of this research is beyond the scope of this paper; however, Horne and Meyer [2004] discuss some work on visualization tools and exploration methodologies for ABM results, and others [Kornhauser, 2009; Kornhauser & Wilensky, 2009] has been developing a novel interactive exploration tool and associated visualization techniques (see Figure 2.1). However, there are several persistent problems with approaches that rely on human knowledge to guide the exploration process. Humans may be biased (either consciously or unconsciously) when exploring the model's behavior, humans can easily make erroneous assumptions about the interaction of model parameters and their

effects on overall model behavior, and bugs (either conceptual flaws or simple coding errors that researchers are unaware of) may be present in the model. All of these factors can lead humans to neglect the exploration of regions of the space that they *expect* to be uninteresting, and therefore miss important findings. Additionally, manual model exploration can be very time consuming and possibly tedious for humans, whereas algorithmic methods can tirelessly explore the space and eventually report findings for a human to review. As a general prescription, I believe that the best practice is to combine human-interactive exploration with computer-automated methods: use human intelligence and intuition to explore the model's behavior, but also use unbiased[2] algorithmic methods to find points of interest that the humans might have missed.

A second (and more automated) approach to model exploration stems from the classic "design of experiments" (DOE) literature [Fisher, 1971], which is concerned with how to efficiently sample points in a space in order to understand the effects of factors in an experiment. DOE has often been employed in agriculture to design experiments to test different growing conditions. For example, suppose there are 10 different chemicals that could be applied to the soil (10 factors), each with three different levels of potency (3 levels), giving a total of 3^{10} possible treatment combinations. Some examples of experimental designs include the factorial design (which tries all levels of all factors), the Latin hypercube design (which guarantees a certain degree of representativeness while sparsely sampling the space – see Figure 2.2), and the sphere-packing design (which attempts to efficiently cover the space while sampling few points.) While these methods are potentially useful for designing

[2]Technically, all search methods apart from uniform random search are biased in some sense, as they make implicit assumptions about the structure of the search space - for instance, that good solutions are more likely to be found near other good solutions. However, computer search methods are arguably not biased in the same way that humans are (this point will be revisited in Chapter 11), and certainly one would not accuse a computer of having an externally motivated agenda when performing model exploration!

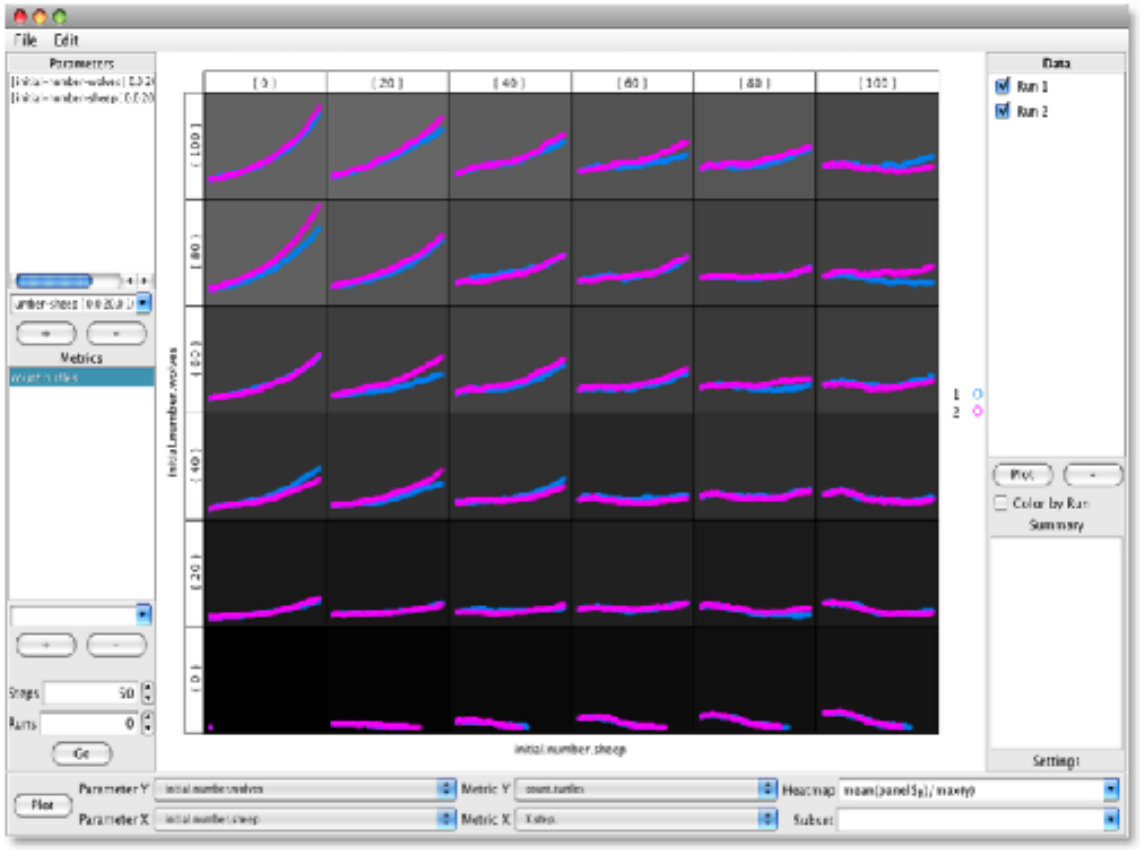


Figure 2.1. Screenshot of a prototype version of Kornhauser and Wilensky's [2009] tool for visual (human-driven) exploration of ABM parameter spaces.

experiments for exploring agent-based models, Sanchez and Lucas [2002] point out several drawbacks to applying classic DOE methods to agent-based simulation. For instance, classic DOE methods often assume that interactions between factors are either linear or low-order effects, which may not be true in ABMs. Also, DOE tends to choose all the points to evaluate ahead of time, whereas in computer simulations it is often possible (and desirable) to choose new points to evaluate sequentially, using information gained from previous results. Naturally, there is considerable interest and ongoing research in applying and extending DOE methods to better handle computer simulation [Kleijnen, Sanchez, Lucas, & Cioppa, 2005], agent-based simulation in particular [Sanchez & Lucas, 2002], and also on "adaptive

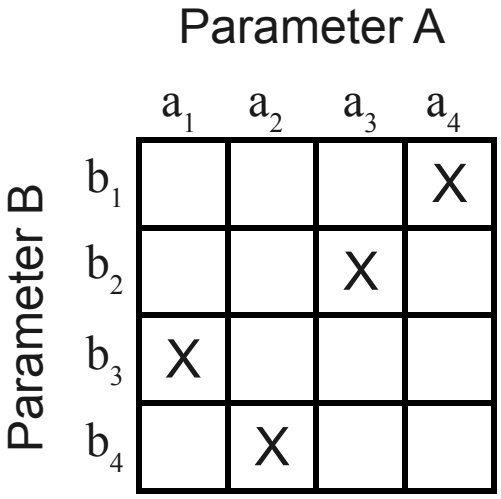


Figure 2.2. Latin Hypercube Sampling (example shown above for a two-dimensional space) samples each parameter setting exactly once within each dimension, as opposed to a factorial experiment design which would sample all combinations of all parameter settings. If interactions between parameters were linear, one might be able to extrapolate behavior across the parameter space from this small sampling. However, ABM parameter spaces are often fraught with complex nonlinear interactions.

designs" (alternatively called "sequential" or "dynamical" experimental designs) [Van Beers & Kleijnen, 2008; Ankenman, Nelson, & Staum, 2008], including application to multi-agent systems in particular [Klein, Bourjot, & Chevrier, 2005]. Recent work has also attempted to use machine learning methods to create inverse mappings between model parameters and measurable model outcomes, based on experiment data [Miner, 2010; Miner & desJardins, 2008]. Using techniques such as these, it is possible to create metamodels of the space (linear, polynomial, kriging, or others), perform sensitivity analysis, or screen out which model factors appear to be relatively unimportant. In the DOE approach to exploring the parameter-space of agent-based models, the focus is usually to say something general about the whole space. In contrast, the genetic algorithms approach discussed in this thesis will

focus more on searching for specific points in the space in order to answer questions about model behavior. However, the distinction between search techniques and adaptive experimental designs can be fuzzy at times, as they sometimes seek to solve similar problems.

Finally, some simulations permit another form of exploration - through proof and analytic methods. If the simulation rules are constrained to be of a particular form (e.g., certain discrete event simulations), it may be possible to use logical inference methods to prove patterns or constraints regarding model behavior without running the model repeatedly with different parameter settings. However, this approach to exploration does not apply to unconstrained agent-based models of complex systems that are written in Turing-complete languages (such as NetLogo [Wilensky, 1999] or Java).

2.2. Search-Based Exploration of ABMs

Search methods provide another way of exploring the parameter space of an ABM. In this context, the word "search" is closely tied to methods of "optimization", since we may design an objective function that expresses the characteristic behavior that we are searching for. As mentioned in the aircraft boarding example in Section 1.3, it is possible to construct objective functions that will search the parameter-space for various types of model behavior or outcomes. However, it can be challenging to design an appropriate objective function that both captures the desired model behavior and provides a good search gradient. One contribution of this thesis is the development of a methodological framework to support the design of objective functions for different model exploration and analysis tasks, which is covered in Chapter 3.

Since the search problem is posed as an optimization problem (maximizing or minimizing the objective function), I should note that there has naturally been considerable research

attending to the optimization of computer simulations in general, partially because there is considerable commercial incentive to finding optimal (and/or robust) configurations of supply chains [Shapiro, 2001], engineering design [Fox, 1971], financial market portfolios [Rockafellar & Uryasev, 2000], and other systems modeled in industry. Numerous optimization techniques have been developed for both restricted and unrestricted problem domains, linear and nonlinear, constrained and unconstrained, with discrete and continuous parameters, for local and global optimization, etc., etc. The academic field of optimization is broad, diverse, and somewhat fragmented. Nevertheless, it is worth mentioning several papers from the simulation optimization research community that do not use evolutionary search, including a review paper [Kleijnen & Wan, 2007] that discusses OptQUEST [Glover, Kelly, & Laguna, 1996], which is a notable commercial package that uses scatter search and tabu search in conjunction with a neural network surrogate model for doing simulation optimization. Wakeland et al. [2005] give an example of using both OptQUEST and a genetic algorithm for doing verification and validation of a software process simulation model. There is also some promising recent work on a new search technique (COMPASS) that provably converges to local optima despite noise under certain conditions [L. J. Hong & Nelson, 2006]. Although this field of literature deals more generally with the optimization of computer simulations, and not specifically with the sub-genre of agent-based modeling, the methods and challenges discussed are often relevant to ABM as well.

My thesis research focuses on genetic algorithms because they possess several characteristics that are useful for this domain. First, genetic algorithms are a metaheuristic technique that is general enough to handle the mix of boolean, integer, discrete, continuous, and categorical parameters that may be present in agent-based models. This rules out gradient-descent-based methods (which require a differentiable function), as well as other

techniques (including linear programming, quadratic programming, integer programming, nonlinear programming, and others) that are tailored for all-numerical parameters or explicit numeric functions. Second, the objective functions are almost always stochastic and may be non-convex[3] and genetic algorithms have often proven effective at escaping local optima in the search space [K. A. De Jong, 1975], as well as progressing towards a goal despite noisy environments [Fitzpatrick & Grefenstette, 1988]. Third, the choice of genetic algorithms is motivated by an intuition that the crossover operator will be able to take advantage of building blocks (in this case, subsets of the model parameters which work together to elicit certain model behaviors), or partial solutions, to speed the search process.

Using *evolution-inspired* search methods to optimize computer simulation parameters is not a new idea – in fact, evolutionary strategies were invented by Rechenberg in the 1960s with engineering-related parameter optimization problems in mind [Rechenberg, 1973], and genetic algorithms were proposed for the parameter optimization of complex systems by De Jong [1980] as early as 1980. Moving beyond a focus on *optimization*, in the 1990s Bankes [1994] proposed that evolution-inspired algorithms be used more broadly for exploring computer simulations and Miller [1998] recommended their use for testing/calibrating/analyzing system dynamics models.

Let us now shift the discussion to parameter search that is specifically in the context of ABM, rather than computer simulation in general. As mentioned in the introduction, most prior work on the use of genetic algorithms for searching the parameter-space of agent-based models has focused on answering specific questions for specific agent-based models. For example, Heppenstall et al. [2007] show that a GA can be an effective method for

[3]In "non-convex" functions, local extrema are not guaranteed to be global extrema. Non-convexity excludes the use of a large class of efficient optimization algorithms, generally making the search task more difficult.

calibrating parameter settings for an ABM of petrol retail markets. Similarly, Caporale et al. [2009] use evolutionary programming[4] to calibrate a model of contagion in multi-national financial markets. Skolicki et al. [2008] employ a *co-evolutionary* algorithm to evolve both terrorist and security scenarios. There have also been several recent examples of using multi-objective genetic algorithms to search for parameters that provide good trade-offs between several output metrics regarding the behavior of ABMs such as trading in financial markets [Rogers, Tessin, & Eurobios, 2004] and emergency response planning [Narzisi et al., 2006]. In general, these projects focused on the specific application area, and not on providing a broader perspective about how genetic or other evolutionary algorithms can be used for ABM parameter search.

One exception to this trend is the recent work of Calvez and Hutzler [2005], which proposes a framework for using genetic algorithms to tune the parameters of ABM. However, even here, their framework was illustrated only by one realization of a genetic algorithm being applied to answer several questions about one particular agent-based model. Specifically, they report preliminary results for one case study of using a GA to optimize different output quantities on the NetLogo Ants Model [Wilensky, 1997a], which simulates an ant colony performing pheromone-based food foraging. However, their case study involved the evolution of just two parameters, they did not compare their results to a baseline measure, and nor did they provide comparisons to any other search methods or different agent-based models. Additionally, the conceptual framework they proposed leaves much room for improvement. The example fitness (objective) functions they describe cover only a subset of the possible

[4]Evolutionary Programming (EP) is an optimization method developed by L. Fogel [1966] that is similar to genetic algorithms although generally lacks the crossover operator. For an explanation of the (sometimes subtle) differences between genetic algorithms (GA), evolutionary strategies (ES), and evolutionary programming (EP), please refer to this overview paper by Bäck [1993].

use cases for model exploration, and they only begin to address the interaction between simulation stochasticity and search algorithm dynamics. They include examples of maximizing the ant foraging efficiency, finding parameters that will yield three concurrent ant lines, and calibrating the model to match predetermined data. In short, Calvez and Hutzler provided a valuable preliminary foray into this research area, but both the framework and methods prescribed deserve considerable expansion, as well as additional validation and support.

To expand on their work, I have developed a more complete/unified framework (elaborated in Chapter 3 for both developing behavioral measures for ABMs and applying them as fitness functions. The framework includes additional use cases such as model testing, sensitivity analysis, identification of critical/leverage points, finding robust or volatile parameter settings, and extreme scenario discovery. This work is partially inspired by John Miller's [1998] seminal work on "active nonlinear testing", which proposed the use of nonlinear search methods for a variety of useful tasks involving the testing and analysis of simulations. Specifically, Miller used a genetic algorithm and a random-mutation hill-climber to search through the parameter space of system dynamics models (SDMs), which model aggregate-level quantities by numerically integrating differential equations over time. Because ABMs and SDMs share several common features, some of these ideas will transfer directly. However, there are differences as well, such as the micro-macro link, and the stochasticity of results that are typical of ABMs, but not present in SDMs. Unfortunately, to date little work has been done to extend or expand upon the ideas that Miller proposed. We remedy this primarily in Chapter 3, as well as with additional considerations interspersed throughout the case studies (Chapters 4 through 7).

In other related work, Brueckner and Parunak [2003] suggested a method of exploring the parameter-spaces of ABMs by using an agent-based approach (one might consider it a

meta-level ABM), coupled with a fitness function to measure how interesting each point in the space was considered to be. In this scheme, which they call *Adaptive Parameter Sweep Environment* (APSE), "Searcher" agents (using heuristics to move through the parameter space) would allocate more trials in areas of higher fitness (provided they had not already been extensively examined). This approach is also reminiscent of Particle Swarm Optimization [Kennedy, Eberhart, et al., 1995] (with global attractive forces between agents based on fitness). In this work, Brueckner and Parunak [2003] demonstrated that their APSE approach could successfully discover phase transitions in one example ABM of distributed graph coloring, and that this approach was more efficient than a grid-based (factorial) simulation experiment. However, they did not provide any comparison with other possible techniques or metaheuristic search algorithms such as GAs.

Yahja and Carley [Yahja & Carley, 2006] have also approached the problem of model exploration and validation, but their method uses a knowledge-based inference engine and causal reasoning, in an attempt to emulate the causal reasoning that human scientists use. Yahja and Carley argue for the superiority of this method over genetic algorithms, but they did not perform any objective comparisons. While it may be tempting to assume that knowledge-level reasoning mechanisms will offer improvements over an evolutionary approach, this is not necessarily the case. It is worth considering that in nature, the "blind watchmaker" has designed many creative and powerful solutions to challenging problems that have, as yet, escaped the ingenuity of human engineers. Regardless of this quasi-philosophical debate, until the capabilities of genetic algorithms have been more fully investigated, they should not be discounted as an effective problem-solving method in this domain.

The list of related work would be incomplete without mentioning the SADDE methodology [Sierra, Sabater, Augusti, & Garcia, 2004], in which a modeler starts with an EBM

(Equation-Based Model – for literature comparing ABM and EBM, see [Parunak, Savit, & Riolo, 1998; Wilensky & Reisman, 2006, 1998]) describing aggregate-level patterns of the system, and then designs an ABM to match the EBM, using evolutionary algorithms for the calibration of ABM parameters. Sierra et al. [2004] demonstrate the SADDE methodology using a case study of the U.S. electricity market. They first develop equations to characterize aggregate-level behavior of the market, and subsequently build an agent-based model of the market. Using genetic algorithms to tune several parameters that affect the behavior of producer and consumer agents in the ABM, they were able to find parameter settings that fulfilled the macro-level requirements, as specified by their EBM. This example again suggests that genetic algorithms may be a useful tool for this type of parameter search task, but this is merely a single data point, and no comparisons were made with alternative techniques.

As an additional side note, there are many other ways that evolutionary algorithms can be combined with agent-based modeling, in addition to searching parameter spaces. In fact, evolutionary algorithms can themselves be conceived of as a multi-agent system. For example, Socha and Kisiel-Dorohinicki [2002] propose a new multi-object evolutionary algorithm using an agent-based design approach, and Stonedahl and Rand [2008b] propose an agent-based model of the diffusion of innovation across social networks, which may equivalently be understood as a distributed genetic algorithm with a restrictive breeding network. Alternatively, the rules by which individual agents act can also be evolved by genetic programming, either before or during the model runs (e.g., [Panait & Luke, 2004]). Also, each individual agent can simulate human decision-making or intelligence by using a genetic algorithm as a mechanism for choosing good strategies (e.g., [Rand & Sondahl, 2004]). I mention these

other possibilities only to distinguish them from the form of ABM/GA integration that my research will focus on, which is parameter-space exploration.

2.3. Related Genetic Algorithm Research

In this section, I will discuss literature regarding genetic algorithms and/or metaheuristic search, and with regard to various challenges that are characteristic of ABM exploration problem domain, even if they are not fully specific to it.

These challenges include the slowness of fitness evaluation, the noise in the fitness function created by model stochasticity, and the issues of chromosomal representation when mixing continuous and discrete parameters.

2.3.1. Noisy fitness functions

One of the challenges posed by searching the parameter-space of agent-based models is the stochastic nature of the simulation. The use of randomness is such a prevalent feature in ABM that NetLogo's agent scheduling is randomized by default and special measures must be taken to cause the agents to always take action in the same order. Randomness commonly plays many other roles in an ABM as well: breaking ties when choosing between alternatives, allowing agents to act based on probabilities. This stochasticity is often a beneficial trait from a modeler's perspective, since it makes the model more robust against fluke events and accidentally biased results. Even if we presume that the phenomena being modeled does not involve any "truly" random processes, the use of numbers drawn from random distributions can serve to represent any source of variance in the system that is not being explicitly modeled. For instance, in Schelling's [1969] classic model of spatial segregation in residential areas (see also the NetLogo version of this model, [Wilensky, 1997d]), when

agents are "unhappy" with their current location, they use randomness to select a new place of residence. Obviously, the real reasons people choose a house are far from random: they may wish to be close to their work, in a good school district, near a park/library, or aspects of the house itself (fireplace, patio, etc) may appeal to them. However, rather than trying to explicitly model heterogeneous agent preferences, many of which are unobservable, for a multitude of criteria along myriad dimensions, the choice of relocation can be simplified to a single decision made randomly. The randomness represents our lack of specific knowledge, and abstracts away many details that are unnecessary to the larger point that Schelling sought to make (i.e. that even "weak" prejudice among individuals can lead to strong "segregation" at the population level).

However, from a search-based perspective, the use of randomness means that even when a model is run with precisely the same set of parameters, different results will occur each time, as a result of different initial seeds for the pseudorandom number generator. A common approach is to treat each model run as a signal with some amount of noise, and to run the model for a fixed number of trials and take the mean (or perhaps median) value of the results. There are two potential issues with this.

- If you are searching the parameter space for a particular objective function, and you have limited computational resources, it is preferable to distribute trials based on how promising the individual (or perhaps the region of the parameter space) is. For instance, if after 10 trials, the fitness of an individual is significantly less than the fitness of other individuals in the population, does it still make sense to run another 20 trials on that same individual, in order to improve the statistical significance of the estimate?

- The noise in the output of an agent-based model often means something. It may be a measure of the predictability or robustness of the current parameters. Furthermore, consider that there may be two attractors in the phase space of the system. If the output measure is 0 half the time, and 100 half the time, it may be misleading to condense this to a mean value of 50.

The former point will be discussed in more detail in Chapter 8, and the latter in Chapter 3.

There has been considerable prior research about genetic algorithms in the presence of noise and uncertainty; Jin and Branke [2005] provide a good survey of this area. Also, Jaskowski and Kotlowski [2008] recently offered several approaches for statistically selecting the best individual among the members of the final generation of the GA, when the fitness function is noisy. More specifically, in the context of calibrating parameters of ABM Calvez and Hutzler [2005] considered the problem of noise, and offered a rough guideline that practitioners should first try running many replications at one point to estimate the "error rate" as a function of the number of replications, to get an idea of how many replications should be run to get an error rate of less than, e.g., 5 percent. However, this approach makes an implicit assumption that the error rate is constant throughout the parameter space, and they do not relate how the estimated error rate actually affects the progress of the genetic algorithm towards an optimal solution. Furthermore, they recommend a novel approach of running only a single replication in most generations but running many replications every Nth generation, to get a better estimate of actual fitness. However, they offer neither theoretical nor empirical evidence that this novel approach is superior to more traditional techniques; the issue of controlling the variation of fitness function noise in a generation-dependent manner like this deserves further investigation. In short, the interplay between the stochasticity of

ABM simulations and the genetic algorithm's ability to search noisy fitness landscapes is not fully understood. Without a deeper understanding and improved methodology, practitioners may choose methods of dealing with noise that are simplistic, ad hoc, and inefficient.

2.3.2. Computational cost

Another challenge for applying genetic algorithms to agent-based model exploration is the typical slowness of fitness evaluation. Agent-based models are often computationally demanding, with thousands of agents interacting over many thousands of model "ticks" (units of simulated time). If the model takes a long time to run and allotted search-time is constrained, then sacrifices must be made which will affect the genetic algorithms performance: either the GA population must consist of fewer individuals, the number of generations that the GA is allowed to run must be reduced, or the individuals must be evaluated less frequently or less extensively. Too small a population will be unable to support sufficient diversity during the search process, and result in considerable "inbreeding", which is likely to prevent any positive effects of crossover. On the other hand, running for too few generations will not allow the GA to converge on a good solution. As has just been discussed in Section 2.3.1, one way to evaluate individuals less extensively is by taking the average of fewer model runs, which constitutes a trade-off between signal noise and computational effort. There is some evidence that the performance of the genetic algorithm on noisy problems can be improved by decreasing the amount of sampling of individuals, while instead increasing the population size [Fitzpatrick & Grefenstette, 1988].

There are a number of additional approaches that attempt to reducing computational time for fitness functions that are slow to evaluate, including using parallel or distributed genetic algorithms, fitness approximation, fitness inheritance, and fitness caching. While

parallel genetic algorithms, surveyed in [Cantu-Paz, 1998], are undoubtedly useful, and will become increasingly so as the world shifts toward more parallel computing architectures, in some sense these approaches are sidestepping the issue problem by applying more hardware. Fitness approximation, surveyed in [Jin, 2005], involves the use of a surrogate fitness function which is cheaper to evaluate than the real fitness function, but provides only approximate results. The genetic algorithm then uses the surrogate fitness function some of the time, and the real fitness function some of the time. This is a promising direction, although automatically finding a surrogate function that gives a good approximation to the result of the complex nonlinear processes that take place in an agent-based model may be prohibitively challenging. (Although surrogate fitness functions fall outside the scope of the present work, some of the analysis of ABM fitness landscapes in Chapter 9 may offer clues about the feasibility of this approach moving forward.) In fitness inheritance [R. Smith, Dike, & Stegmann, 1995], individuals in the population sometimes inherit fitness values by averaging the values of their parents, rather than evaluating the real fitness function. This is based on the premise that children will tend to have fitness values that are highly correlated with their parents fitness, and so we may be able to approximate directly from their parents (some fraction of the time) without actually evaluating them. However, initial studies using fitness inheritance were performed using simplistic fitness functions, and there is concern that the approach does not scale well to non-convex functions found in real-world problems [Ducheyne, De Baets, & De Wulf, 2003]. On the other hand, fitness caching [Kratica, 1999] is a straightforward approach that involves the memoization of the fitness function, so that future evaluations at the same location in the parameter space will not have to run the ABM again. The more that individuals are likely to recur during the search, the more savings will result from caching. However, previous cases of fitness caching [Kratica, 1999;

Kratica, Tosic, Filipovic, Ljubic, et al., 2001] did not consider the interplay between caching and *noisy* fitness functions, which may negate the result mentioned above [Fitzpatrick & Grefenstette, 1988] that increased population sizes are often more beneficial than increased sampling. As part of this thesis work, I provide both analytic and empirical investigations of fitness caching in the presence of noise, found in Chapters 8 and 9. Besides attempting to eliminate the computational cost of re-running ABMs at the same settings, there may be value in recording all fitness evaluations made during the progress of the genetic algorithm, either to use this information to more intelligently guide the search process or to post-process this data to discover other interesting features of the parameter space.

2.3.3. Chromosomal representations

As demonstrated by the Ethnocentrism model [Axelrod & Hammond, 2003; Wilensky & Rand, 2003] introduced in Chapter 1 (see Figure 1.1), as well as a recent linguistic model about how language change may diffuse in a social network (see Figure 2.3), agent-based models may contain a variety of parameter types, including boolean parameter, integer-valued parameters, discrete numeric parameters, continuous numeric parameters, categorical parameters, strings of text, and potentially even more unusual types such as lists, matrices, colors, or bitmap images. In section 2.2, I mentioned that this was one of the motivations for using genetic algorithms, since they are a flexible enough search technique to handle these different representations. Unfortunately, there is scarce literature studying tradeoffs and appropriate choices of GA chromosomal representation when dealing with mixed data types. Although Holland's original discussion of genetic algorithms [1975] mentioned the possibility of using non-binary alphabets, the majority of early GA research focused on binary strings. After all, on a digital computer, any data type can always be encoded as a sequence of bits.

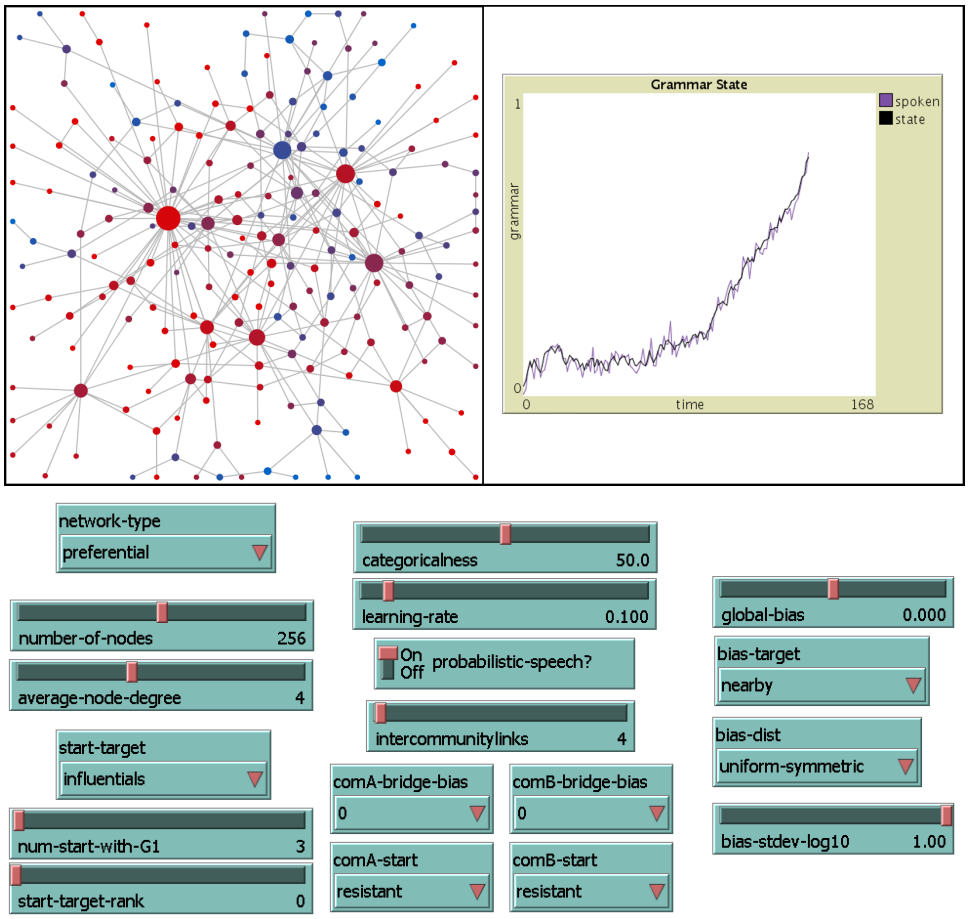


Figure 2.3. *Top left:* A visualization from the Diffusion of Language NetLogo model, which investigates language change occurring in a social network context. *Top right:* A plot of average population-level grammar preferences versus time (demonstrating complex dynamics). *Bottom:* The 18 controlling parameters of this model: 8 categorical, 4 integer-valued, 5 real-valued, and 1 boolean.

On the other hand, the independently conceived evolutionary strategies [Rechenberg, 1973] used only real-coded genes from the start. The first implementation of real-coded genes in the "Michigan school" of genetic algorithms was in Weinberg's 1970 thesis. In time, the field of real-coded genetic algorithms blossomed and there has been much more work in this area than can be overviewed in this document. Some debate around these two gene representations arose, as well as attempts to understand their strengths and weaknesses,

and resolve the theoretical differences between them [D. E. Goldberg, 1991]. Additionally, there is continued debate on the topic of crossover on real-coded genomes, as naive averaging approaches proved to be inadequate in many cases. Various mechanisms have been proposed, including attempts to apply insights from binary-coded crossover [Deb & Agrawal, 1995], as well as recent work on "parent-centric" crossover [Ballester & Carter, 2004a] which in some sense functions as a self-adaptive mutation rate, based on the spread of the population in the space.

Despite the volume of discussion about real and binary representations, as noted above, there seems to be very little literature regarding the appropriate use of mixed representations (though a hybrid approach was employed recently with reasonable results [Gantovnik, Anderson-Cook, Gürdal, & Watson, 2003]). Although this topic is not a particular focus of this dissertation, throughout this work I have used a variety of binary, real-valued, and mixed representations. Anecdotally, I did not find a strong trend regarding which chromosomal representation might be superior for this domain, as the genetic algorithm appeared to work well with several representations. However, there are many open and intriguing questions, including: when is it preferable to convert parameters into binary format as opposed to letting them remain numeric? how should crossover be performed in a mixed-coding representation? and how can mutation-rates be appropriately calibrated across the mixed-representation genome? I feel that a more rigorous treatment of these questions would make a good topic for further research (possibly even another thesis) in this area. Answering these questions would be useful in the particular domain of ABM exploration, but perhaps also offer insight into the use of mixed genome representations in general. Lacking definitive answers from the literature, I find the mixed representation to be most natural, as this most

closely preserves the genotype-phenotype mapping, and this was the representation used for the benchmark comparison performed in Chapter 9.

2.3.4. Exploration versus optimization

Considerable research has focused on genetic algorithms as function optimizers, seeking input values that will achieve the single best global maxima or minima (for a given objective or fitness function) [Ashlock, 2006]. However, as has been argued by De Jong [1993], genetic algorithms were designed *not* as function optimizers, but rather as a mechanism inspired by real evolution, which engages in an adaptive exploration process of a complex and time-varying landscape. This observation provides further support for the choice of genetic algorithms, as similarly the goal may in part be exploration of the parameter space, rather than finding a single optimal point. While modelers may be interested in global minimization or maximization, this is not the extent of their curiosity. For example, knowing the point of the parameter space that produces the least smog in model of air pollution is useful, but in some cases identifying multiple regions of low smog is arguably more beneficial. Thus diversity maintenance in the genetic algorithm's population may be a more important consideration than convergence to a global optima in the search-algorithm. More generally, it is worth emphasizing that exploration does not equal optimization, though the two tasks are related, and often have substantial overlap.

Some work which bears more toward exploration, is the recently proposed "scouting algorithm" [Pfaffmann & Zauner, 2001], which is essentially an evolutionary algorithm hunting for "surprises", or points in the search space that give a value considerably different from what is expected, given the values at nearby neighboring locations in the space. It has also been proposed that the idea of "scouting" can be integrated with an evolutionary algorithm

that is searching for some objective function, as a means of avoiding premature convergence on local optima [Pfaffmann, Bousmalis, & Colombano, 2004; Bousmalis, Hayes, & Pfaffmann, 2007]. Another interesting approach to encourage exploration in the space is the use of *co-evolution*, inspired partially by the "estimation-exploration" algorithm proposed by Bongard and Lipson [2005]. In their algorithm, however, the goal is to identify a system that would produce the same results as some unknown system, which can be probed at great expense. However, a similar coevolutionary approach could employ a population of metamodels that are attempting to estimate the fitness space, and award individuals that give results that were poorly predicted by the metamodels, thus indicating a region of potential interest.

2.3.5. Searching for a cheap lunch

Another broad consideration which must be addressed is whether the central thesis problem is well-posed. In particular, this thesis is about demonstrating the efficacy of genetic algorithms for the exploration of agent-based model parameter-spaces. Part of this process involves demonstrating that genetic algorithms are more effective for this than other search techniques. However, the space of all possible agent-based models covers much territory, and as agent-based models can be very different from each other, the shape of their parameter-spaces may also vary greatly. Is it, therefore, reasonable to assert that some search algorithm is better than any other for this large class of problems? This objection stems from the aptly (and entertainingly) named "No Free Lunch theorem" [Wolpert & Macready, 1997], which in paraphrase, states that over the class of all possible functions, no search technique will outperform any other, and in particular none will perform better than a random search. Of course, no one is ever trying to search the space of "all possible functions", so this theorem's pragmatic application is limited; real world problems come in various shapes and sizes, but

their search spaces often contain regularities and continuities that distinguish them from an arbitrary randomly-chosen function. However, there have been further arguments made that there is still "no free lunch" even for more restricted classes of functions with features that are similar to some real-world problem domains [Droste, Jansen, & Wegener, 2002].

I have several responses to these concerns. First, the philosophical/hypothetical objections are not very constructive; from a practical standpoint, there is an important need for tools to explore behavior in agent-based models, and in order to build such tools, some search algorithm (or ensemble of search algorithms) must be chosen. There is a tendency for people to over-interpret the *No Free Lunch* theorem as being more damaging to meta-heuristic search research than it is in practice; in some situations the NFL theorem can help prove the general superiority of one search technique over another. For example, in an ironic twist, Whitley [1999] applied the NFL theorem to show how one form of search-space encoding (gray encoding) could in fact be provably superior to another for most real-world problems. Second, I should clarify that we do not expect one single search algorithm (genetic or otherwise) to be the most efficient mechanism for exploring every conceivable agent-based model. However, it may be that some single method does work quite well for exploring most agent-based models. Third, despite the many differences between agent-based models, there are a number of features that most agent-based models share, and some search algorithms will certainly be better suited to these features than others. For instance, the stochasticity of model runs results in noise in the objective function, and some search methods handle noise more effectively than others. In fact, as we will show in Chapter 9, genetic algorithms prove to be broadly effective for parameter exploration on a representative set of models and exploratory tasks (see Chapter 9).

The No Free Lunch theorem provides a cautionary warning to search and optimization researchers, that one should not expend fruitless energy seeking the single silver bullet or holy grail that will solve every problem. However, it does not diminish the importance of seeking effective search algorithms for reasonably broad categories of real-world problems, nor is it a serious impediment to showing that genetic algorithms can be highly effective for ABM exploration tasks, which is a goal of this thesis.

2.4. Tools for Automated ABM Search and Exploration

This section will attempt to detail past research, and the present state of the art, with regards to tools for ABM parameter-space exploration. In recent years there has been considerable growth in the area of toolkits and libraries that support the creation of agent-based models, but considerably less emphasis on supporting exploration and analysis of those models [N. Gilbert & Bankes, 2002; S. C. Bankes, 2002]. As mentioned in section 2.1, there has been some recent work on visualization and interactive exploration tools [Kornhauser, 2009; Horne & Meyer, 2004]. However, I will restrict my attention here to search-based exploration, as that is most directly relevant.

A recent survey of extant agent-based modeling toolkits [Railsback, Lytinen, & Jackson, 2006] reviewed four major toolkits suitable for research modeling: Swarm [Minar, Burkhart, Langton, & Askenazi, 1996], MASON [Luke, Cioffi-Revilla, Panait, & Sullivan, 2004], Repast [North, Howe, Collier, & Vos, 2005; Collier & Sallach, 2001], and NetLogo [Wilensky, 1999; Tisue & Wilensky, 2004]. Swarm models may be written in either Java or Objective-C, and modelers wishing to perform parameter search could write their own routines from scratch in one of these languages. MASON models are written in Java, and MASON was designed to integrate with ECJ [Luke, 2000], an evolutionary computation library written in Java

by one of the authors of MASON. Similarly, Repast S includes a partial framework for parameter optimization, which users can extend Java classes and interfaces to create their own algorithms of moving through the search space. Implementations of hill climbing and simulated annealing are provided, but there is no built-in support for genetic algorithms[5]. Of these ABM toolkits, NetLogo is the only toolkit that has its own multi-agent language and integrated modeling environment. NetLogo also provides a Java controlling API, so that users could write their own programs for doing exploration, as well as an interface that allows NetLogo to be controlled by Mathematica, which has several built-in optimization algorithms. However, the current available methods for parameter search generally require substantial computer programming and demand a fluency with code libraries or in some cases writing algorithms from scratch. They are largely inaccessible to modelers who are not advanced programmers, and even for advanced programmers, they do not provide any built-in scaffolding of the most common tasks (e.g., searching for a large mean value across some number of replicate runs). Nor is there straightforward support for using parameter search to perform more complex search tasks such as model calibration, sensitivity analysis, volatility or robustness testing, or phase transition discovery.

MASS (Multi-Agent Simulation Suite) [Iványi, Bocsi, Gulyás, Kozma, & Legendi, 2007] is a relatively recent addition to the ABM toolkit world. It contains a formal (though somewhat constrained) modeling language (FABLE), but it also includes a model exploration module (MEME) [Iványi, Gulyás, Bocsi, Szemes, & Mészáros, 2007] which is capable of interfacing with models written in Repast, NetLogo, or plain Java, as well as FABLE models.

[5]Repast S does include a genetic algorithms library as part of the distribution, but it is configured for evolving agents (or agent-level properties), and not for model parameter-search.

MEME is the most similar tool to *BehaviorSearch* that is currently available. It provides flexible support for several different experimental designs (such as Latin Hypercube Sampling), and is currently moving toward support for more dynamic or adaptive search algorithms for experimentation with model parameters. For the past few releases, MEME has included an option that mentions John Miller's ANT (active non-linear testing) functionality, and a very recent build of the MEME software added a plugin for genetic algorithms. Neither of these features is documented, and they appear to be relatively experimental at this point.However, this is clearly a direction that this tool is moving towards in the future, and MEME should provide a second tool (besides *BehaviorSearch*) that will support aspects of the query-based model exploration framework discussed in Chapter 3. There are several important differences between MEME and *BehaviorSearch*. At present *BehaviorSearch* only interfaces with the NetLogo modeling toolkit, whereas MEME is designed to support models written with a variety of toolkits. However, as discussed further in Chapter 10, *BehaviorSearch* (like NetLogo) has an explicit goal of being *low-threshold* – that is, being easy for novice modelers to use and get started with, and its simple/straightforward integration with NetLogo supports that goal. (*BehaviorSearch* also provides a host of features useful to support advanced users, as detailed in Chapter 10).

In addition to these five well-known toolkits discussed above, there many alternatives. Nikolai and Madey [2009] provide a fairly exhaustive list of extant agent-based modeling libraries and tools (53 in total), though the strength of their survey is breadth rather than depth. I will only discuss a few other tools/projects which seem particularly relevant.

- The SeSAm environment [Klügl, Herrler, & Fehler, 2006] also provides tools for visual modeling and experimenting with agent-based simulation. While the project

website makes a reference to support for searching for parameters that maximize some objective function, I was unable to discover any such facilities, either in the software's user interface or documentation. This appears to be a work that is still in progress.

- Pfaffmann and Jenkins [2008] expressed the intention to begin designing tools and/or techniques to automate experimentation with agent-based models, using some form of evolutionary search techniques (such as scouting [Pfaffmann et al., 2004]), but this project does not appear to have produced any publicly-available tools thus far, and the status of the project is currently unknown.
- Yahja and Carley [Yahja & Carley, 2006] have been developing WIZER, a "what-if analyzer" for the purpose of exploring and/or validating very large social agent simulations (specifically, the BioWar [Carley et al., 2006] simulation of bioterrorism attacks). As was mentioned in Section 2.2, this tool employs a logic-based inference and causal reasoning engine (not genetic algorithms) to navigate through the model's parameter space. While they propose their methodology as a general approach, the WIZER software is currently designed to work specifically with the BioWar model and would require considerable modification to transition it into a generic software tool that is applicable to typical agent-based models. Furthermore, the emphasis of their work is on designing a system that can perform experiments and reason independently, rather than providing a tool for modelers to use to explore their own models.

In conclusion, there is currently an unfilled need for a *low-threshold* tool for performing parameter search for agent-based model exploration.

CHAPTER 3

Query-Based Model Exploration: A Theoretical Framework

"Science is what we understand well enough to explain to a computer. Art is everything else we do."

– Donald Knuth

"The purpose of models is not to fit the data but to sharpen the questions."

– Samuel Karlin

By Knuth's definition (and arguably many others), the process of developing agent-based models is currently more of an *art* than a *science*. While the artifacts of the process (constructed models) are specified at a level of specificity that a computer can understand and execute, the *act of modeling* is a complex product of human ingenuity. I believe this state of affairs will remain largely the case until the advent of significant breakthroughs in artificial intelligence. However, this does not mean that certain aspects of the modeling process cannot become more scientific, or even automated by computers. This chapter introduces a theoretical framework for exploring agent-based model behavior using evolutionary search algorithms. In doing so, it is my goal to contribute toward the *science* of agent-based model analysis.

The description of the Query-Based Model Exploration (QBME) framework provided in this chapter in split into four major sections:

- The first section (Section 3.1) formalizes the concept of the behavior of an ABM in terms of the data it produces.
- The second section (Section 3.2) focuses on the formulation of measures of agent-based model behavior.
- The third section (Section 3.3) focuses on ways that these measures can be applied to accomplish a variety of important model analysis tasks.
- The fourth section (Section 3.4) discusses genetic algorithms, and how behavioral measures are used as fitness functions in the search process.

The QBME framework hinges on a paradigm shift in how people explore and analyze agent-based models. I will briefly explain this paradigm shift, and then go on to discuss the two organizing principles that the framework is built around – *levels of analysis* and *diversity*.

Parameters and Paradigms

When you run an agent-based model, you are implicitly asking (and answering) the question: what is the behavior of the model given parameter settings $(p_1, p_2, p_3, ...)$. As we'll discuss more below, you may have to run the model multiple times with these parameter settings to get a good idea of what the model's behavior is, but this is the general paradigm. I would like to encourage a paradigm shift for model analysis, which essentially inverts the question. Instead of asking "what behavior will I get with a certain set of parameters?", what if instead we were asking questions of the form "what parameter settings will give me a certain behavior?" More formally, if we are interested in some behavior B, then what settings of the parameters $(p_1, p_2, p_3, ...)$ will result in the greatest expression of B. The difference in model exploration workflow for these paradigms is shown in Figure 3.1. While

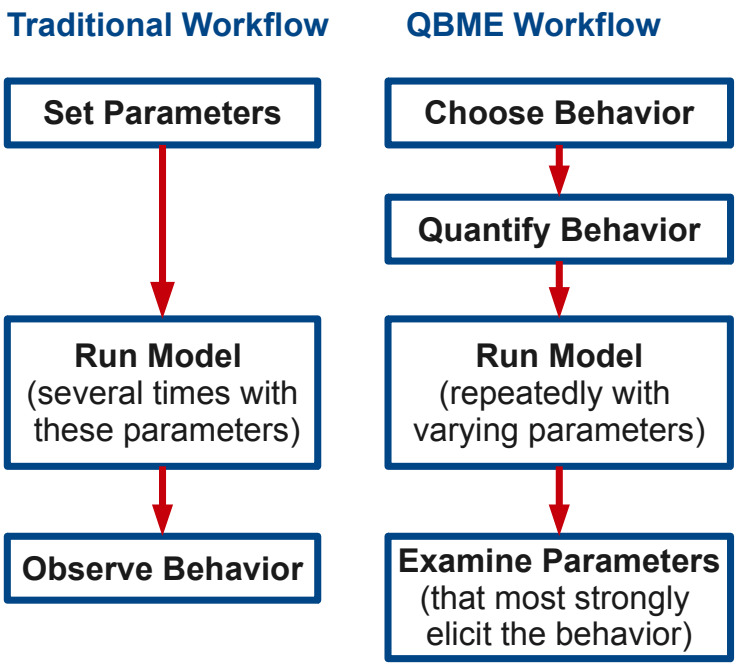


Figure 3.1. Flowchart highlighting the difference between the QBME paradigm and the traditional paradigm for model exploration.

the idea of shifting perspective in this way is very simple, the new paradigm requires the elucidation of several steps of this process.

(1) How can we quantify an interesting model behavior in a way such that we can search for parameters that yield it? (Discussed all throughout Sections 3.2 and 3.3.)

(2) How can we efficiently and effectively search the parameter space for a given behavior? (Discussed in Sections 3.4.1 to 3.4.4 .)

(3) How should we interpret the parameters that are returned by the search process? (Discussed especially in Sections 3.4.5 and 3.4.6.)

This chapter will address these questions while elaborating the QBME framework, and the following chapters will address them more concretely in the context of case studies.

We move next to a brief introduction of the framework's organizing principles (*levels of analysis* and *diversity*), although these themes will be revisited continually throughout this chapter.

Thinking in levels

In the context of learning about and understanding complex systems and multi-agent models of those complex systems, the importance of thinking about *levels of complexity* has been well-established by a bulwark of research in the cognitive and learning sciences [Wilensky & Resnick, 1999; Penner, 2000; Raia, 2005; Chi, 2005; Jacobson & Wilensky, 2006; Sabelli, 2006; Goldstone & Wilensky, 2008; Levy & Wilensky, 2008b]. The importance of levels has also been recognized by complex systems and ABM methodologists [Bar-Yam, 1997; Parunak et al., 1998; Epstein, 1999; Wilensky & Rand, in press], as well as dissected and discussed by various philosophers of science[1] [Simon, 1973; Wimsatt et al., 1994]. In most cases, the focus of this work has been on two levels of analysis: the agent (or individual) level, and the aggregate (or population) level. These two levels are at the heart of understanding complex systems because the agent-based model demonstrates how aggregate-level behavior emerges from the interaction of individuals. However, in order to provide a framework for creating behavioral measures of agent-based models, we must consider levels of analysis that are both below (i.e. intra-agent) and above (e.g., comparing multiple simulation trials) these standard levels[2]. The QBME framework also must take the temporal aspects of agent-based

[1]For example, the philosophical subfield called "mereology" is dedicated to the formal study of the logical properties of the relation of part and whole. However, whether the existing literature in mereology can substantially contribute to our understanding of emergence in complex systems remains in doubt.

[2]Not all of these "levels" will necessarily meet the definition of "emergent levels" as set out in [Wilensky & Resnick, 1999] – although phenomena and patterns at a higher level often emerge from interactions at a lower level, in some cases I will use the word "level" to denote simple containment relationships, or hierarchical levels.

simulation into account, in order to characterize model behavior *over time*, which defines a "temporal level" of analysis (if we allow the word "level" to be broadly construed). When unpacking the levels of analysis that are required for the exploration of the parameter spaces of agent-based models, we will begin at the most basic level (that of a single agent), and work our way up to the highest level (diversity of results among different search methods).

Diversity in complex systems

Diversity is the second organizing principle for this framework. In recent years, there has been an increased interest in studying the effects of diversity in complex systems [S. Page, 2010; S. E. Page, 2008; L. Hong & Page, 2004; Eagle, Macy, & Claxton, 2010; Santos, Santos, & Pacheco, 2008]. In most cases, these studies focus on diversity at the individual level – to what extent do agents in a population differ from one another, and what effect does this variation have on system-level behavior or performance outcomes? However, in the context of this framework, I will interpret the term *diversity* inclusively, to refer to any form of variation (or difference) at any level. Thus construed, diversity is a broad but powerful theme, and methods of quantifying diversity can by applied at each of the *levels of analysis* discussed in this framework, resulting in a rich collection of behavioral measures. Depending on the level of application, these measures can capture concepts of homogeneity versus heterogeneity, similarity versus difference, constancy versus change, predictability versus unpredictability, and sensitivity versus robustness. We will find below that diversity is not a general enough concept to capture all behavioral measures that we might be interested in, but it does give us considerable purchase as an organizing principle for this framework.

3.1. Formalizing ABM Behavior

Before discussing various types of behavioral measures, it will be helpful to unpack what is meant by *ABM behavior*, and introduce some formal notation to increase the rigor of the discussion that follows. An agent-based model is fully specified, in some sense[3], by the computer program which is to be executed, along with a conceptual correspondence (which may be loose) between entities in the model and entities in the target phenomena being modeled. However, it is only through *running* the agent-based model that we can discover the model's behavior - not through static inspection of the model's source code. In the general case, an agent-based model is a computer program written in a Turing-complete language, and may implement algorithms of arbitrary complexity. Thus, from a theoretical perspective, *halting problem* style arguments apply, proving that behavior cannot be determined without running the code. From a more empirical perspective, individual agent rules are often fairly simple algorithms, but the aggregate patterns resulting from them are quite complex, and in practice even advanced modelers tend to have considerable difficulty predicting emergent model behavior. While methods might be developed in the future that can characterize model behavior without model execution, especially for limited subclasses of models, I am not optimistic about this general approach. In any case, at present, model behavior is best characterized by the output of executing the model (simulation).

In addition to the source code, an agent-based model has some associated set of input parameters designated by the model author as variables of interest. We are specifically interested in formalizing the following: what is the behavior of the model for a given choice

[3]One might insist that to *fully* specify the model, one would also need a complete specification of the interpreter/compiler for the language the model is written in, as well as the computer hardware which runs it, etc. But this line of thought eventually leads to philosophical considerations which are beyond the scope of this thesis...

of parameters settings p in the space P of all possible parameter settings. A specific p represents bindings $(p_1, p_2, p_3, ...)$ for each parameter of the model. As an illustrative example, to make the formal notation we are building more accessible, let us consider the Wolf Sheep Predation model [Wilensky, 1997e], which is a fairly simple ABM of predator-prey dynamics in a closed ecosystem. (Although this is an abstract model, it bears a striking resemblance to real-world predator-prey relationships, such as those recorded between wolves and moose that on the relatively closed ecosystem of Isle Royale, in Michigan, U.S.A. [Peterson, 1999].) The model's interface, including the model parameters exposed by the author, is shown in Figure 3.2. In this case, p_1 would correspond to the show-energy? parameter, p_2 to the grass? parameter, p_3 to the grass-regrowth-rate parameter, p_4 to the initial-number-sheep parameter, and so on. (Actually, one of these parameters (show-energy?) only affects the visualization of the model, and does not change the model behavior; although varying visualization options can be crucial for humans to explore and understand model behavior, we will exclude such parameters from consideration in the exploration and analysis process.)

Agent-based models almost always contain stochastic elements (if only in the initialization of agent properties, or in the randomized scheduling of agent behavior). Thus, when executing a single simulation of the model, in addition to the parameter configuration p, we must also specify a seed $\phi \in \mathbb{Z}$ for the pseudo-random number generator[4] that the simulation will use as a source for randomness. An agent-based model can thus be viewed as a function $m : P \times \mathbb{Z} \rightarrow B$, where B is space of model behaviors for a single simulation of the model.

[4]In practice, the RNG seeds must be selected from a large, but finite range, rather than from the set of all integers. For example, in NetLogo, there are approximately 1.8×10^{16} choices for a random seed. Since this far exceeds the amount of behavioral sampling that one can afford to do, the finiteness of seed choices is immaterial in practice.

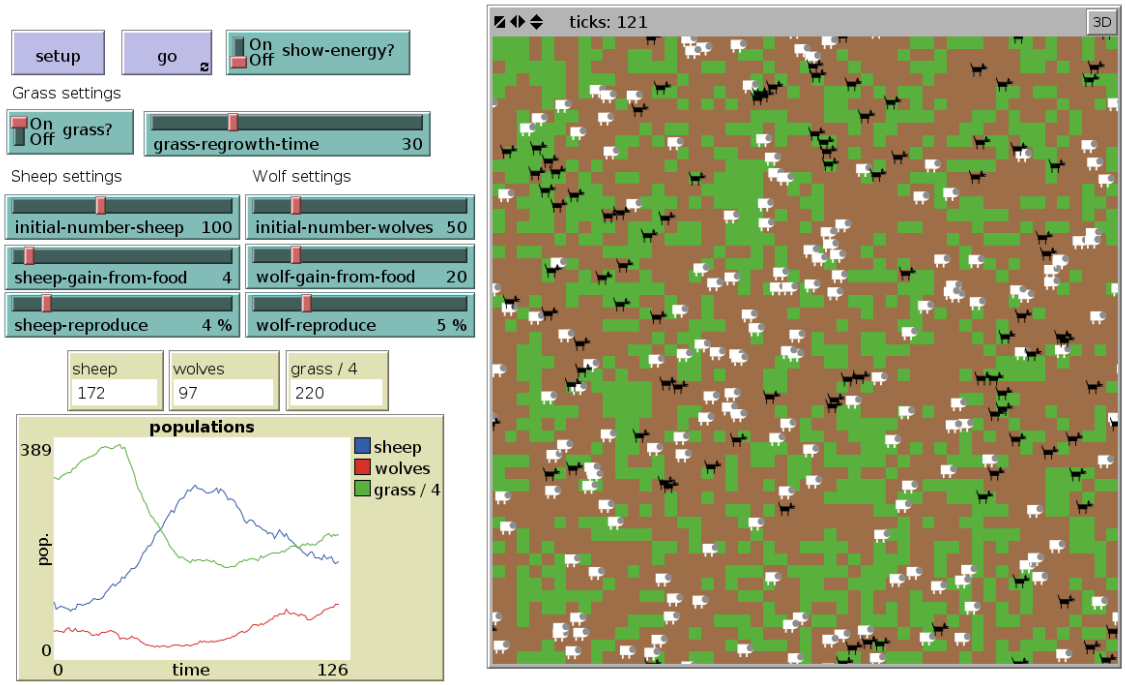


Figure 3.2. The interface of NetLogo's Wolf Sheep Predation model [Wilensky, 1997e]. Model parameters are shown on the left, along with several model outputs, such as the current number of sheep, wolves, and grass, and a plot of these values over time. The model view on the right shows the spatial locations of the mobile agents in this model, which are (unsurprisingly) wolves and sheep, as well as the amount of grass present on each stationary patch agent.

In other words, for any given parameter setting $p \in P$, starting with random seed ϕ we will obtain some model behavior $b \in B$.

This conception is fairly straightforward, but it requires the full specification of B, which is more complicated. Since agent-based models produce a vast quantity of data, we will approach this from the bottom up. The state of a single agent a_j at a single moment (t) in simulated model time may be specified by the set of agent-level variables (v_k). In addition to agents, agent-based models also consist of an environment e, and the state of the environment

may be captured by some number of environmental variables[5]: $e_1, e_2, ..., e_l, ..., e_L$. Thus for a model, a complete state of world with J agents, may be specified by the ordered tuple $w = (e_1, e_2, ..., e_L, a_1, a_2, ..., a_J)$, where each a_j is itself an ordered tuple of agent-level variables $(v_1, v_2, ..., v_K)$ where the number of variables K may vary between different types of agents. Assuming simplest case where all agents have a uniform number of agent-level variables, and where each variable is a scalar value that requires β bits of information to store it, this description of the world requires $\beta(JK + L)$ bits of information. In reality, agent-level and environmental variables may sometimes be more complex data structures such as lists, trees, or matrices, which may be dynamically resized over time. For instance, it is not too uncommon for agents to accrue a list of information gathered from previous time steps, which the agents may use when making decisions, as is the case in the El Farol model [Rand & Wilensky, 2007; Rand & Sondahl, 2004], based on earlier work on boundedly rational agents [D. Fogel, Chellapilla, & Angeline, 1999; Arthur, 1994]. The previous expression was to describe the world at a given time t. However, to describe the behavior of an agent-based model, it is usually insufficient to consider only the state of the model at the end of the simulation, or any other static snapshot of model state. The behavior of an agent-based model unfolds over time, and thus we must consider the state of the world (w) at each time t. We can fully characterize the model behavior for a single simulation as $b = (w_0, w_1, w_2, ..., w_t, ..., w_T)$, where T is the maximum number of steps the simulation is run.

However, as we mentioned above, agent-based models tend to have stochastic elements, meaning that the way the simulation unfolds will depend on the random seed ϕ. Since it is impossible to run the model for all values of $\phi \in \mathbb{Z}$, in practice we choose a finite subset of

[5]In NetLogo, these would correspond to "observer"/"global" variables. NetLogo's patches, although often used to model the spatial environment, are in fact agents themselves, and their state is captured in the list of agents a_j.

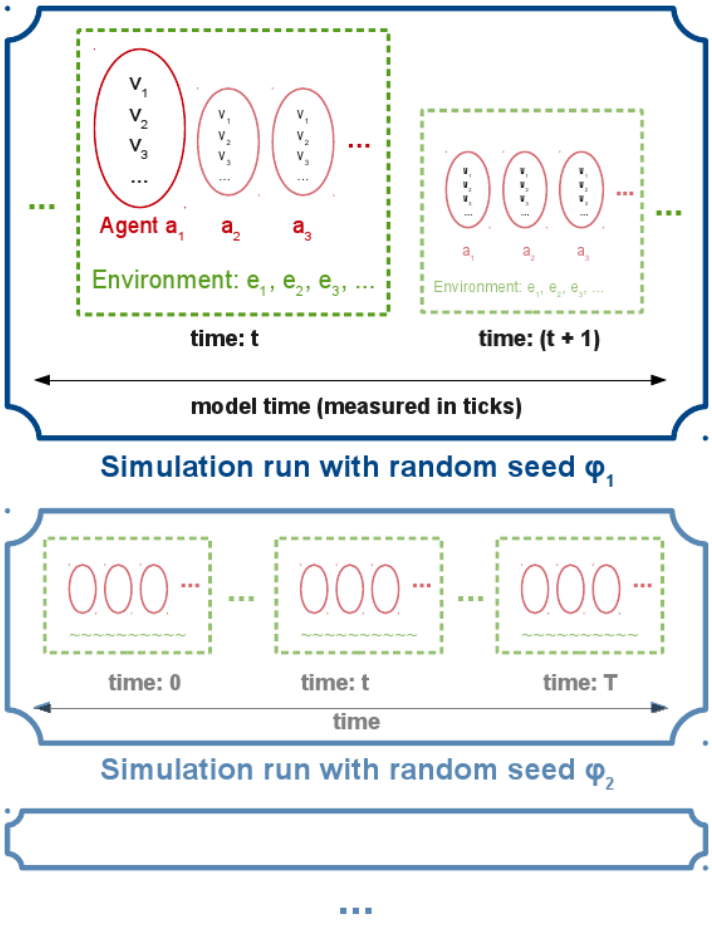


Figure 3.3. Diagram illustrating the state information required to capture the behavior of an agent based model for a given set of parameter settings.

distinct random seeds $\Phi \subset \mathbb{Z}$, with $|\Phi|$ large enough that one an obtain a reasonable estimate of the distribution of single-run behaviors. In this text we will refer to $|\Phi|$ as either the "number of sampling repetitions" or "number of replicate runs". Thus, for a fixed parameter configuration p, the behavior of the model is fully specified by the complete trajectory across time for each of the agent-level variables of each agent, and their environment, for a repeated set of model runs with varying random seeds. This characterization of ABM behavior is illustrated in Figure 3.3. The total amount of memory required to capture this behavior is

$\beta|\Phi|T(JI + L)$. As an illustrative example, let's consider a reasonably simple ABM such as the Wolf Sheep Predation model:

(1) There are several thousand "patch" agents representing square sections of ground (on which grass may grow), and which each store two numeric values that change over time.

(2) There are varying numbers of wolf and sheep agents (but let's estimate an average of 200) which each store four numeric values which change over time.

(3) There are no global variables which require tracking, thus $L = 0$, and $(JI + L)$ is roughly 6000

(4) a simulation run might go for just 1000 (T) ticks, and we might run as few as 30 ($|\Phi|$) replicates with different random seeds.

(5) Each numeric value requires 64 (β) bits of memory.

In this case, the total amount of memory required to store the behavior of the model is about $1.3GB$, and that is just for a one set of parameter settings p. However, there's a deeper underlying problem here. Even if we had ample storage space to keep a full description of the simulation behavior, it would be too detailed to be useful. As with Lewis Carrol's fictitious map that had "the scale of a mile to the mile" (and Jorge Luis Borges' elaboration on this theme in the short story *On Exactitude in Science*), our characterization of the behavior of a model by recording the complete state space of the model is less than enlightening. Like a good map (or a good model[6]), a good measure of model behavior must strip away (or condense) almost all of the detail, so that only a concise characterization of behavior remains, which is focused on the type of patterns that the modeler (or analyst) is interested in

[6]Rosenblueth & Wiener [1945] provide additional discussion about the appropriateness of detail level in scientific models.

investigating. However, the complete characterization of behavior is still important, because it forms the starting point for developing more concise (and also more useful) measures. From a theoretical perspective, all other measures of model behavior may be defined as functions that reduce or compress the complete state information described above. In practice, the data is filtered and condensed concurrently while the simulation runs, avoiding the collection and storage of large amounts of unnecessary data.

In order to apply the QBME methodology using evolutionary algorithms (or other meta-heuristic search), we require a behavioral measure that yields a single numeric value[7], and that value should quantify how well the model does (or does not) exhibit the behavior we are interested in exploring, for a given parameter settings. Formally, we must construct a fitness function $f : P \rightarrow \mathbb{R}$, where $f(p)$ expresses the extent to which parameter settings p cause the model to exhibit a specific behavior $b* \in B$. This function may be decomposed into two stages: $f(p) = f_2(f_1(p))$. The first stage is $f_1 : P \rightarrow B^{|\Phi|}$, which is obtained by running the model with various parameter settings and random seeds – $f_1(p) = (m(p, \phi_1), m(p, \phi_2), ..., m(p, \phi_{|\Phi|}))$. The second stage requires condensing this vast amount of data into a single number: $f_2 : B^{|\Phi|} \rightarrow \mathbb{R}$. This second stage function may itself, be the composition of a variety of smaller "ingredient" functions, which are behavioral measures that collapse or condense data across various dimensions or aspects of the behavioral space. These ingredient measures are the topic of the following section, Section 3.2.

[7]This is not strictly true. While this thesis focuses on the use of single objective functions, it is both possible and desirable to extend the QBME framework to multi-objective search, wherein the search algorithm attempts to maximize/minimize multiple objective functions simultaneously.

3.2. Formulating Measures

3.2.1. Intra-Agent Measures

As noted above, individual agents within a multi-agent model each have a collection of properties, or agent-level variables. In the NetLogo language, there is a combination of built-in variables along with user-specified agent-owned variables (see Figure 3.4 for an example of agent-level variables for a single sheep in the NetLogo's Wolf Sheep Predation model [Wilensky, 1997e]).

While it is possible to create a measure using any conceivable function of these variables, the most common measurement at this level is just to extract the value of a single agent property: e.g., an agent's energy level of that sheep. Short arithmetic expressions (sums, products, differences, and ratios) of agent properties can also useful, such as the product of an agent's mass and its velocity in a physics-based simulation. In this example, the resulting measure would represent the momentum of the agent – a quantity which is not directly stored for any agent, but can be easily computed from a combination of two other agent-level properties.

Although measurements of diversity and similarity are less suited to the individual level of analysis than they are to the upper levels (as we will discuss below), they can still arise and be useful in certain contexts. For example, at a multi-agent modeling workshop that I recently led, one of the participants was interested in modeling changes in people's political alignment and affiliation. In this case, each agent's x-coordinate (xcor) would represent a range between conservative and progressive on issue 1 (perhaps a social policy issue) while the agent's y-coordinate (ycor) would represent a range on the same scale for issue 2 (perhaps a fiscal policy issue). Moreover, this scheme can be extended to any number of dimensions,

Figure 3.4. An "agent-monitor" (or "inspector") window in NetLogo provides a listing of agent-level variables, along with the current values of each variable, for a single sheep in NetLogo's Wolf Sheep Predation model [Wilensky, 1997e]. In this case, all but the last variable (energy) are default/built-in variables that every mobile agent ("turtle") in NetLogo possesses. The energy variable is an additional user-defined variable specific to the Wolf Sheep Predation model.

wherein an agent would have K agent-level variables v_1, ..., v_k, ..., v_K corresponding to alignment with K different issues. In this case, measuring the homogeneity of v_k values within a single agent would provide a measure of political consistency and/or strict party affiliation, whereas heterogeneous values might indicate more complex ideological affiliations, or more independence in the subscription to political institutions. Intra-agent measures form natural building blocks for creating multi-agent measures at the next level of complexity.

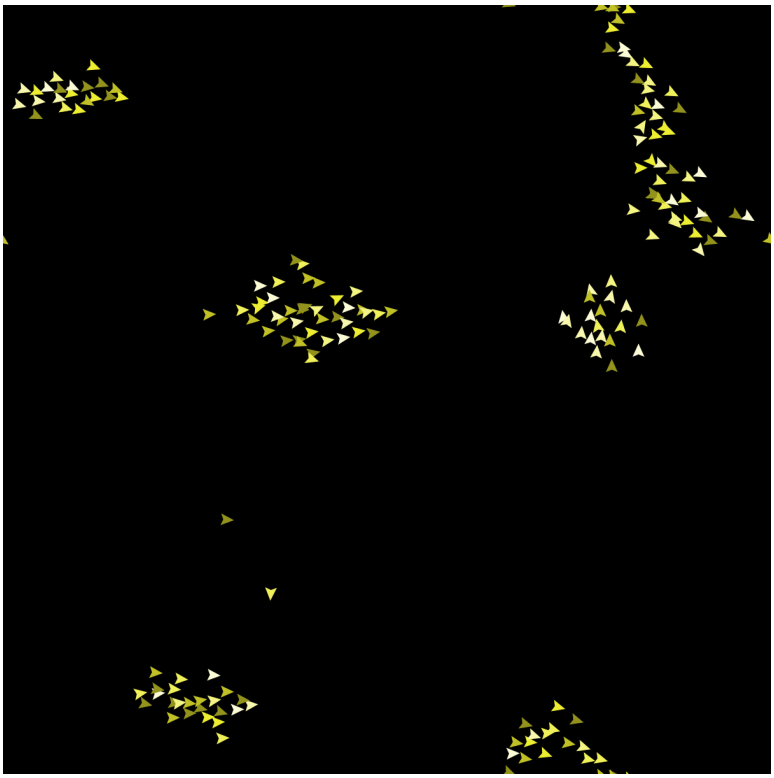


Figure 3.5. A visualization from NetLogo's Flocking model [Wilensky, 1998], after the birds have self-organized into a number of disparate flocks.

3.2.2. Agent Group Measures

It is not uncommon for ABMs to contain several distinct types of agents (NetLogo uses the concept of "breeds" for this purpose). An example of this would be a predator-prey model, where predators and prey each belong to a distinct agent type (as depicted by the wolf and sheep shapes in Figure 3.2). Even when there is only one type of agent, it may be logical to divide the agents into groups based on some criteria (e.g., geographical location). For example, in the Flocking model [Wilensky, 1998], birds self-organize into groups/flocks (see Figure 3.5). When such groups exist, we may be interested in computing inter-agent measures among the birds in each flock (in fact, a fleshed-out example of this appears in the measurement of vee-shaped formations in Chapter 4), rather than treating all birds

as belonging to the same population. In fact, recent research [Levy & Wilensky, 2008a] suggests that the identification of smaller group-level structures (between the individual and aggregate level) may be important for understanding, and thus analyzing, emergent behavior in complex systems. In many cases, though, the whole population of agents in the model is simply considered as a single group.

In each of these cases, it is useful to characterize behavior at the *group level*. In terms of the the QBME framework, this moves the discussion up one *level of analysis*, and accordingly it becomes possible to design measures for different aspects of model behavior than were possible at the single-agent level. Perhaps the simplest group level measure is a count of the number of elements (agents) in the group. This group-based measure requires no information about the agents it is composed of, beyond their mere existence, and thus it essentially ignores the lower agent-level properties. However, many richer group level behavioral measures can be constructed by combining single-agent measures in a variety of ways. One straightforward measure is the "mean" value of some single agent-measure, taken across the group. Depending on context, alternative forms of averaging (median, mode) are also useful, as well as extracting extrema from the group (minimum/maximum value). While there is no limit on the complexity of functions that could be employed to reduce a collection of individual values to a single group value, much leverage can be gained from appealing to the organizing principal of *diversity*. Measures of diversity (such as the standard-deviation of agent-level measures) are surprisingly useful for characterizing group-level phenomena. For instance, a group of agents with diverse set of x and y coordinates is more loosely clustered, while a group of agents with homogeneous coordinates is tightly clustered. In an economics model, measuring variations between the amount of money held by the constituent agents in a population provides important information about how wealth is distributed. There are

also some more specific measures of diversity of wealth (or wealth inequity), such as the Gini coefficient [Gini, 1912], which will reappear in Chapter 5 in the context of measuring the inequity of the degree distribution in networks. Beyond measuring the standard-deviation of the distribution, there are a number of other measures of statistical dispersion, including the variance, range, interquartile range, mean difference, median absolute deviation, and coefficient of variation. Higher order moments about the mean, such as skewness, are potentially useful for measuring the shape of diverse distributions.

Note that in most cases, we consider agent groups to consist of an unordered set of agents, and thus most agent measures are functions of the *distribution* of agent-level measures among members of the set. However, less frequently, the agents within the group may have a natural ordering (e.g., by the age/longevity of the agent, or by position, or by any agent-level measure of arbitrary complexity), which agent group measures may need to take into account. As an example, an agent-group measure in a simulated economic marketplace could look at the correlation between company age and company size, rather than looking at either variable in isolation. We can view correlation between variables as an extension of the theme of diversity – how are the agents' variables similar or different with respect to one another? However, not all behavioral measures fit neatly into the theme of diversity. For instance, another simple agent group measure is to take just a single agent's value from the group, but choose that agent based on certain criteria: e.g., measure the wealth of the largest (or smallest) company in the group. Such measures take this form: Choose an agent based on agent-level measure A, and then report a different agent-level measure B, measured on that agent. This method extends naturally into the more general agent-group measure: Filter the agent group based on agent-level measure A, and then apply *agent-group* measure B to the filtered subset. Finally, as mentioned above with example of groups of flocking birds within

a model, it is possible to have groups of groups, in which case multiple layers of group-level behavioral measures may be applied. Although in principal one could imagine applying group-level measures recursively for many layers, in practice most agent-based models have fewer than two levels of grouping.

3.2.3. Network-Based Measures

Along with the rise of complex systems science, there has been an increasing focus in the field of *network science* [Amaral & Ottino, 2004; Barabasi, 2003; Watts, 2004; Newman, Barabasi, & Watts, 2006; Newman, 2010]. Although network science has roots in both mathematical graph theory (stemming from Euler's famous solution to the Königsberg bridge problem [Euler, 1736], as well as later groundbreaking work by Erdős and Rényi [1959; 1960]) and social network analysis (from Moreno's invention of "sociometry" [Moreno & Jennings, 1938] to later experimental and theoretical work [Milgram, 1967; Granovetter, 1973; Burt, 1995; Wasserman & Faust, 1994], this new (or revitalized) field has a stronger focus on the widespread applicability of networks as a tool for improving understanding across a wide range of disciplines and phenomena (protein-protein interactions, metabolic pathways, gene regulatory networks, neural networks, air transportation, epidemics, scientific and artistic collaborations, ecological food webs, the electrical power grid, the Internet, the world wide web, etc.). (A full introduction to this active area of research is simply not possible in the space available here, so interested readers are encouraged to follow up on the citations provided above.) Agent-based modeling goes hand in hand with the rise of network science, providing a tool that goes beyond static network analysis, and thus allowing modelers to explore dynamic processes on and in networks at both the individual and aggregate levels. The ABM approach also holds promise for gaining insight into not just the patterns, but also

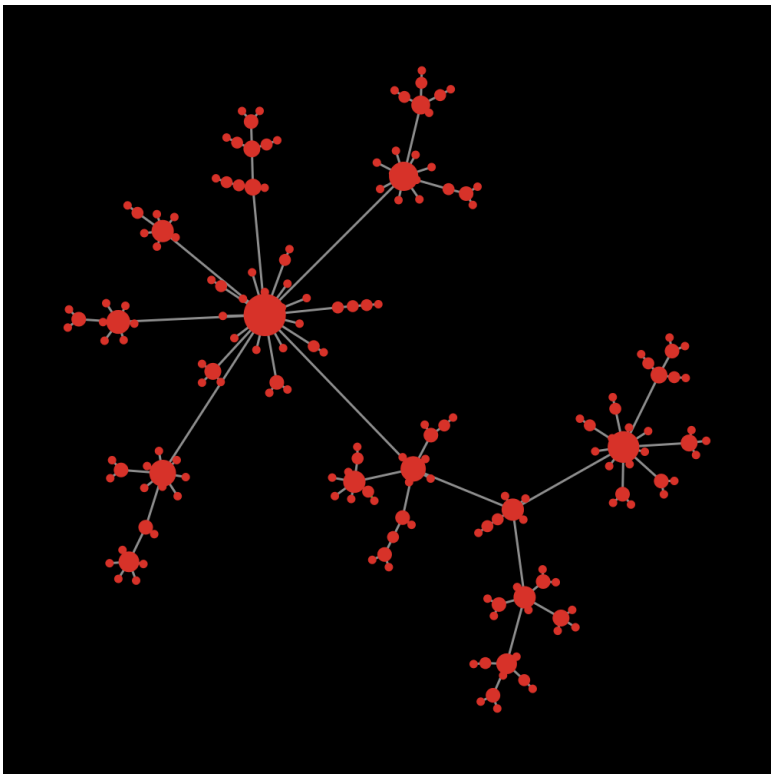


Figure 3.6. A visualization from NetLogo's Preferential Attachment model [Wilensky, 2005], which demonstrates how the power of positive feedback (a "rich get richer" situation) can create power law degree distributions in natural and engineered networks. For scale-free networks, one important measure of the degree distribution is the scaling exponent which describes the power law. For general networks, a measure of how skewed the degree distribution is may help in understanding the network structure.

the "mechanisms" by which networks form and evolve. See Figure 3.6 for a visualization of the NetLogo Preferential Attachment model [Wilensky, 2005], which demonstrates the mechanism Barabasi and Albert [1999] proposed for generating scale-free networks.

In the context of this chapter, the crucial point is that there are many agent-based models where it is insufficient to a characterize agents as simply belong to one group or another (i.e., sorting the agents into different bags, and looking at properties of those bags). Rather, important aspects of model behavior may rest on the connections (links) between

agents, and how these connections are structured in relation to one another. This is precisely what a network perspective provides; the *network level* of analysis exists above the *individual level* but below the *group level*[8].

One difference between group-level and network-level measures, is that whereas most group-level measures (apart from group size/count) tend to be condensing/consolidating some individual-level measure, there are a variety of interesting network-level measures that do not depend on properties of the agents (nodes[9]) Network-based measures can mostly be divided into two types: those are descriptive of a single node, and those that are descriptive of the network as a whole. The former range from simple measures, such as a node's degree (number of links to neighboring nodes), to more complicated measures of a node's position in the whole network, such as a node's betweenness centrality (related to the number of shortest paths from all other nodes that would travel through this node). When network-based measures are applied at the individual level, then some form of *group level* measure is required to aggregate the results. For instance, one might measure the average degree, or the maximum betweenness centrality, among the nodes in the network. In the latter case, the network level measure may apply to the network as a whole. This too can range from simple measures, such as the number of links in the network or the number of connected components, to more complicated measures such as the network diameter (the length of the longest shortest path between any two nodes), chromatic number of the network (how few colors can be used to color the graph such that no neighboring nodes share the same

[8]While it is possible for each node in a network to represent a group of agents, in this case we would probably shift our perspective to consider each network node an individual agent (serving as a proxy for a group).
[9]One unfortunate byproduct of network science's interdisciplinary roots is that the nomenclature is not standardized. In graph theory, a graph is composed of *vertices* and *edges*. In social science, social networks tend to be composed of *actors* and *ties*. In parts of computer science, they are *nodes* and *links*, which are the terms I will generally use.

color), or the number of motifs (small characteristic network structures like triads) which are subnetworks of the network. As a third use of networks, one may use network-based measures to assign agent to different groups, at which point group level measures can be applied. In particular, there has been much interest in recent years in determining community structure for a given network, and various algorithms have been suggested (e.g., [Newman, 2006]) for partitioning the network into disjoint sets, based on the relative link density within and between sets.

This cursory discussion above barely scratches the surface. Networks can be directed or undirected, and weighted or unweighted, simple or non-simple. There are whole areas of algebraic graph theory and dynamical systems devoted to mathematical analysis of graphs and of Markov processes taking place on graphs. Hypergraphs (which connect more than two nodes together with one link) also exist, though they are not commonly used in complex systems and agent-based modeling research at present. However, to avoid leading this discussion too far afield, readers are referred to other readily available sources, such as Newman [2006; 2003; 2010], for more detailed discussion about a variety of network properties and network-based measures. Network-based measured will also reappear in Chapters 5 and 7, although in these cases the measures are being applied within the context of the ABMs, rather than as extrinsically-defined behavioral measures or fitness functions. The main point to remember is that network-based measures provide important tools for quantifying the richly structured information contained in the relationships between agents.

3.2.4. Temporal Measures

Time is a critical element of agent-based models. Unlike certain mathematical techniques that solve for equilibrium solutions or attractor states as $t \rightarrow \infty$, ABMs model time in

discrete units, and may reproduce patterns of behavior that unfold over time. It is a challenging, but necessary task, to quantify these temporal behavioral patterns. Time may be viewed as a higher "level" in the same containment hierarchy as individuals and agents, in the sense that each time period (or "tick" in NetLogo parlance) contains an agent group, which in turn contain individual agents. In this view, the model history is composed of a large number of disconnected snapshots of the model world, each capturing a (simulated) moment frozen in time. Alternatively, it is more flexible to view time as a dimension that is orthogonal to the previous concerns of agent and group measurement. In this way, temporal measures can condense time across agent level, group level, or network level measures. As a specific example, let us consider temporal averaging. In the Wolf Sheep Predation model, each wolf has an energy property. We might want to measure the *average maximum* energy of the wolves or the *maximum average* energy of the wolves. In the first case, for each time t we could compute the maximum energy of the wolves, and then take the average for all values of t; this would tell us generally how energized the currently most vital wolf (which would change over time) was. In the second case, we could first find the average energy across the lifespan of each wolf, and then take the maximum of those values; this would tell us the average energy value of a specific single wolf, which was the most energized over the course of its life. Both of these measures take the average over time, and the maximum across individual agents, but depending on which measure is applied first, the interpretation is slightly different. There are, in fact two other interpretations of the English phrase "maximum average energy", which could involve taking the average across wolves followed by the maximum across time, or the maximum across time followed by the average across wolves. However, the main point was that even when applying the same measures (maximum and

average) at the same levels (group level and time level, respectively), the result can change based on the order of application.

Returning to our theme of "diversity", temporal variation (as measured by standard deviation or variance of an agent or group-level measure over time) is indicative of volatility. In Chapter 4 we will see an example of measuring the variation in flock heading (a group-level measure) over time, to find flocks of birds that have unstable changing headings, rather than converging to a common fixed heading.

Another important point is that unlike group-level measures, where we are condensing a set of results, with temporal measures we are condensing an ordered sequence of data. So while unordered measures such as averages, minimum, maximum, variance, etc, may apply, these neglect important sequential structure. Applying our theme of diversity in a temporally restricted manner, we create measures that examine the difference between the state of the world at time t, and the state of the world at time $t + 1$. In this case, we are brushing up against ideas from calculus: differentials and approximating derivatives. In the Wolf Sheep Predation model, measuring the change in the number of wolves or sheep in the world (a group level measure) over time gives us the birth rates at each point in time. Since applying the difference between each time step from 1 to T still leaves us with $T - 1$ data points, we still need to apply a temporal measure that will collapse across time, such as an average or maximum.

Often, we are only interested in the model behavior close to the end of the run, after the model has had a chance to "settle down" from its early-behavior transient state into its normal steady-state behavior. In general, there is no guarantee that model behavior will ever "settle down", or that steady-state behavior exists. However, it is still often preferable to sample model behavior later on in model time, rather than near the beginning, when

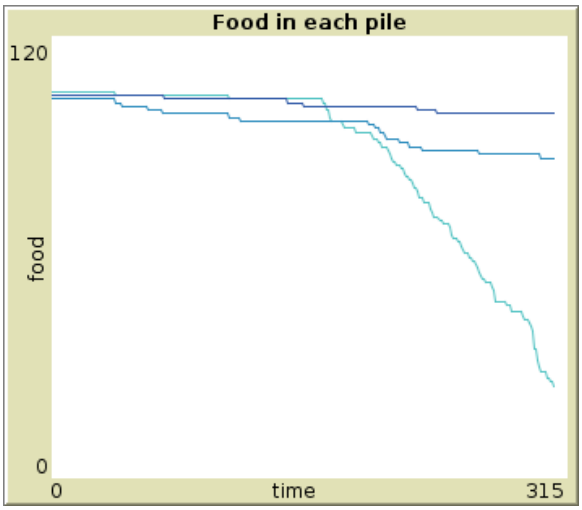


Figure 3.7. Plot of food remaining in each of the three original food source piles, during a typical run of the NetLogo Ants model [Wilensky, 1997a]. The much steeper slope in the decline of one of the three piles corresponds to the presence of a pheremone-based ant trail to that pile, which causes the ants to exploit that food source more quickly.

measures are likely to be overly influenced by random initial conditions. In both exploratory examples in Chapter 4 and the benchmark examples in Chapter 9, there will be examples where behavior is measured only over the last 100 ticks. Although time is an important component of ABMs, there are also many circumstances in which model behavior can be measured simply by taking a measurement of the state of the world at the final tick of the simulation. This is useful if you're only interested in where the simulation ends up, and not the path it took to get there. In this case, the temporal measure simply discards most of the data from the run.

3.2.5. Ant food foraging example

Before moving to the next level of complexity, let's unpack a more detailed example of a measure that ties together ideas from several of the areas described above. This example comes

from the NetLogo Ants model [Wilensky, 1997a], where ant agents interact via pheromone trails they drop in their environment while foraging for food. In the Ants model, there are 3 separate food sources located at varying distances from the colony entrance. Food is modeled as a numeric property of the patch agents that comprise the environment – each patch may store up to 2 units of food. Figure 3.7 shows a plot of the amount of food remaining in each of the 3 food sources (piles). The sharp decline in one of the food piles over time indicates that the ants have been harvesting that pile quickly relative to the others. In this Ants model, one qualitative behavior of interest is the emergent formation of ant lines, which permit much more efficient exploitation of food sources. As a proxy for this qualitative behavior, we can examine create a quantitative measure of the rate of depletion for a specified food source. To be fully specific, let's suppose we are interested in whether an ant line exists to the farthest away food source between 400 and 500 simulated ticks. The intra-agent measure will simply be the food patch-level variable for each patch. The patches that make up each food source will be considered separate groups, and the group-level measurement is a sum of the food patch-level measure across the patches in each group. We will filter the group-level results, so that only the food count for the source that is farthest away remains. At the temporal level, we will take the positive difference of the group-level measure over time for $400 \leq t \leq 500$ (though note that if we are only interested in $t \leq 500$, it is unlikely that we will continue to run the simulation longer, so the condition $t \leq 500$ may be unnecessary). The temporal measure is completed by taking either the maximum or the mean across these time steps. Taking the maximum value would be helpful for detecting whether an ant line ever existed between 400 and 500 ticks (although it could be thrown off if, despite the lack of trail, quite a few ants happened to collect food from that pile at the same time), whereas

taking the mean value would be more indicative of whether an ant line persisted for a large portion of the time between 400 and 500 ticks.

3.2.6. Cross-Replicate Measures

As discussed above, it is necessary to run stochastic models multiple times with different random seeds (replicate runs) in order to evaluate their behavior. The measures we have built up so far (agent level, group level, network level, and temporal) each characterized behavior for a single run. Moving up one more level of analysis, cross-replicate measures combine/condense the results of the single-run measures. Since there is no particular order to the random number generator seeds used for replicate runs, the results from multiple runs form an unordered set of values. Quite commonly we are interested in the behavior of a "typical run" of the model, or perhaps in the "average" or "characteristic" model behavior. In many cases, taking the mean or median of the individual runs will serve this purpose. However, for many models, the idea of a "typical run" turns out to be a myth, as model results may vary widely, and the variation of runs can reveal something more important about the model behavior than any single run would. Again, we appeal to the theme of diversity - this time at the level of diversity among runs, rather than diversity among agents, groups, or agent/group properties over time. As before, the standard deviation of the results provides a rough (but useful) measure of diversity among the runs. A low standard deviation suggests consistency and also predictability of results, whereas a high standard deviation corresponds to less predictability. However, the large diversity of results can stem from a number of possibilities. It could be the result of a process that is simply very noisy, and the outcome is quite random; this is often not very interesting, although it is helpful to know how noisy/random a model can be. On the other hand, the diversity of results may stem from

a *path-dependent* process, where sensitivity to initial conditions can lead the model toward two or more different *attractors* in the model's state-space. This topic will be discussed in more detail below in Section 3.3.2. In either case, the standard deviation will serve as an indicator of diverse results, and the practitioner would be wise to examine the results closely to determine the underlying cause of the variance. Additionally, more complex distributional measures (such as statistical measures of bi-modality) might be developed and used to try to identify specific cases of divergent results. Rather than looking for a typical run or for diversity of results, in situations like model testing (discussed in Section 3.3.6) it can also useful to simply search for extreme model behavior. In this case, taking either the minimum or maximum of the set of replicate results is expedient.

3.2.7. Cross-Parameter Measures

Though cross-replicate measures are often a natural stopping place for measure-building, it is possible to go up yet a further level, and create measures that compare the results of running the model with different parameter settings. This type of measure operates on the output of cross-replicate measures. There is one particular cross-parameter measure that is worth mentioning: measuring the differential with respect to some parameter. The goal here is to measure how much change in behavior (as quantified by some cross-replicate measure) will result from a small change in the value of a model input parameter. More formally, given a cross-replicate measure $m(p)$ for parameter setting $p = (p_1, p_2, ...)$, the cross-parameter differential measure is $\frac{m(p) - m(p - u\delta)}{\delta}$, where u is a unit basis vector of P, corresponding to the specific parameter being varied. As will be discussed below in Section 3.3.3, this measure can be useful for identifying phase transitions or critical thresholds in the model's

parameter space – places where small changes in a parameter can result in large changes in model behavior.

3.2.8. Pattern-Based Measures

There is one more important genre of measure, which does not fit as neatly into hierarchy of levels that we have been discussing, but can be applied at multiple levels. This is the genre of "pattern-based" measures, which compare simulation data to patterns or data that is from some external source (i.e., a source extrinsic to the model). In keeping with the theme of diversity, these measures are quantifying the sameness or difference between a specified *reference pattern* (generally obtained or derived from empirical or real-world data) and the results of the simulation. The relevant question that these measures often seek to answer is: how can one quantify the difference between the reference patterns and the model's behavior? Answering this question is important for both model calibration and sensitivity analysis, as discussed below in Sections 3.3.4 and 3.3.5.

Pattern-based measures can be applied at all of the *level of analysis* we have discussed so far. At the individual level, one can compare whether individual agents match the properties of the object they are representing in the model's target phenomena. For instance, in a model of evolutionary biology, one might compare the percentage of so-called "junk DNA" with the amount empirically measured in the wild. At the group level, the distribution of agent-level measures can be compared against an empirical distribution – e.g., how closely does the simulated distribution of longevity of a species match the observed distribution? It's also possible to look for spatial patterns formed by the group – e.g., in Chapter 4 we measure how closely a group of birds matches a "vee" or "eschelon" formation, such as those exhibited by Canada geese (and certain other species of large birds). At the network level, all kinds of

network measures can be compared against those extracted from real-world networks – e.g., in a model of the growth of online social media networks, how does the average clustering coefficient of the simulated network compare with the clustering coefficient of Twitter? At the temporal level, simulated time series data can be compared with data from longitudinal studies or historical records; more detailed examples of this are discussed in Chapters 6 and 9. At the cross-replicate level, distributions of model results can be compared with distributions of real-world experimental results – e.g., in a model of airplane boarding, the length of time taken to board the craft will vary (both in the real world and in the model), and these distributions of boarding times can be compared. Similarly, at the cross-parameter level, changes in model results resulting from changes in parameters may be compared to results from experiments where parameters were varied. Of course, pattern-based measures can also be used to compare model results with patterns that *have not* (or *have not yet*) been found in the target phenomena. Searching for unrealistic or exotic patterns may sometimes be useful in exploring the range of possible model behaviors, as mentioned in Section 3.3.6 below.

3.2.9. More complicated measure combinations

It is worth mentioning that one may combine the various measures above in many and various ways, in particular with respect to our theme of diversity and change. We may also measure the change (over time) of the uniformity/diversity of a group of agents over time. Or, we might measure the reverse: that is, the uniformity/diversity *of* the change. Beyond measuring the change of some measure, we may look at the change in the change of some measure (i.e., the second derivative). Or rather than looking directly at the pattern-based measure discussed in the previous section, we might look at measures of the *change* in the

pattern-based measure - either over time, or across different parameter settings. And again, instead of looking at the change in the pattern, we might look for the pattern of the change. For example, in the NetLogo Fireflies model [Wilensky, 1997b], fireflies are repeatedly changing state (going from lit to dark, or dark to lit), and then changing number of lit fireflies over time can be plotted. Within this time series, we can seek patterns of change, possibly using tools from signal processing, including autocorrelation, spectral decomposition, Fourier/wavelet analysis, etc. These combinations are not intended to be an exhaustive list, but merely demonstrate some of the rich possibilities for using diversity/change as a theme for constructing more complicated measures from the simpler measures discussed above. However, often simple measures work quite well for driving the model exploration process, and more complicated measures should only be used when the target model behavior is sufficiently complicated to merit them.

3.3. Application of Measures to Model Analysis Tasks

Now that we have discussed the various ingredients and building blocks for developing measure to characterize and quantify behavior in ABMs, let us turn our attention to the import question of how these measures can be applied to important modeling analysis tasks. This portion of the QBME framework builds on the earlier ANT (Active Nonlinear Testing) framework proposed by Miller [1998], with appropriate expansions for the context of agent-based modeling (as opposed to the equation-based system dynamics context, where ANT was originally introduced).

3.3.1. General exploration

Before moving on to various specific model analysis tasks/applications, I would like to make a case for general exploration. That is, rather than having a concrete task/goal in mind, it is useful for modelers to simply to plumb the depths of what is possible in a model's behavior. Modelers commonly explore model behavior in their own ways - twiddling the parameters of the model, trying different combinations, running the model multiple times and looking at what happens. Modelers might describe this activity simply as "tinkering", "playing with the model", or "getting a feel for things". This informal tinkering process is a natural (and I would argue necessary) part of both model development and model understanding. Through tinkering, modelers are probing the system to learn something about it. It is an iterative process of developing small hypotheses (or perhaps merely "hunches") about model behavior, and testing them. I would like to argue for augmenting this tinkering activity with additional QBME-enhanced tinkering. The QBME framework provides tools to get a feel for model behavior from the opposite direction of standard tinkering. In standard tinkering, each time someone runs the model with certain parameter settings, they are trying to answer the question "what model behavior do these parameter settings produce?" The QBME framework allows one to formulate inverse questions of the form: "what parameter settings will produce this kind of model behavior?" Admittedly, because the QBME approach requires the use of a GA (or other search algorithm) that must run the model a large number of times with different parameter settings, this new approach to tinkering does not provide as short of an interactive feedback loop for exploring the model as standard tinkering. I have three responses to that:

(1) the power to answer inverse style questions about model behavior makes it worth the wait.

(2) the approaches are complementary - while waiting for the results of a QBME query, one can continue to tinker with the model in the old-fashioned way, and the two tinkering methods will mutually inform exploration directions for the other.

(3) the development of new software tools (e.g., see Chapter 10) along with the rise of massively parallel cluster/grid/cloud computing platforms will continue to shorten the time and effort required to issue QBME queries and receive results.

One may make the additional objection that it can be difficult for modelers to create behavioral measures to search for. To this objection, I have four responses:

(1) Modelers are already creating a variety of numeric measures of model behavior, for the purposes of analysis and visualization. Many (though not all[10]) of these measures will be suitable for driving an exploratory search process.

(2) While certain behaviors can be very challenging to quantify, there are many other interesting measures of a model's behavior that are very straightforward to formulate, and can still provide new insight into model dynamics.

(3) My hope is that the creation of frameworks (such as QBME) will provide modelers with the grounding they require to start constructing useful measures for query-based exploration.

(4) Like many other skills needed for good modeling, some practice will be needed to gain fluency using QBME methodology. I believe that the learning overhead will be well worth the gain.

[10]Besides being a good measure of the target behavior, a measure must also satisfy additional criteria in order to promote efficient exploration of the search space. This will be explained further in Section 3.4.2.

Using ABMs of collective animal motion as a case study, Chapter 4 will provide several examples of using QBME for general exploration purposes, to discover interesting behavior that can be elicited from the model, and the parameter settings which lead to that behavior.

3.3.2. Path dependence, multiple attractors, and state space analysis

In Section 3.2.6 above, we mentioned that the diversity of results from different replicate model runs can stem from a *path dependent* process [Brown et al., 2005; Arthur, 1998], where sensitivity to initial conditions can lead the model toward two or more different *attractors* in the model's state-space. Thus, we can use measures of variance among runs as a mechanism for detecting path dependent behavior: situations where the model can arrive at qualitatively different outcomes depending on chance events and random initial conditions. Sometimes places in the model's parameter space with multiple attractors are indicative of a *phase transition* occurring in one (or more) of the model's parameters. This general approach to phase transition identification was taken by Brueckner and Parunak [2003] in their multi-agent parameter exploration method. Although it is a good indicator, the presence of multiple attractors does not always indicate a phase transition, and we will discuss an alternative approach to phase transition identification in Section 3.3.3 below.

As a concrete example, two potential *attractors* in the Wolf Sheep Predation model are "complete extinction" and "sheep inherit the earth"[11] [Wilensky & Reisman, 2006]. In the "complete extinction" case, the sheep die out first, causing all the wolves to perish. In the "sheep inherit the earth" case, the wolves die out first, leaving the sheep to multiply endlessly (assuming there is no constraint on the sheep's supply of grass). Consider a measure of the

[11]There are other attractor states of the system, such as where the wolf and sheep populations oscillate in a stable cyclic pattern, but these two extreme attractors best illustrate the point I wish to make about variance of results.

number of sheep after 400 ticks. With the model's default parameter settings (which starts with just 100 initial sheep), running the model 10 times produces these 10 values: 0 0 0 0 5148 14924 21310 32215 71078 78892. The four 0s represent the "complete extinction" attractor, while the other values likely represent the "sheep inherit the earth" attractor (at various stages of the sheep population's exponential climb toward exhausting the computer's available RAM).

3.3.3. Phase transitions, critical thresholds, and leverage points

In Section 3.2.7 above, we mentioned the usefulness of cross-parameter measures for locating phase transitions in the model's parameter space. To elaborate on this theme, let us consider a specific example, the NetLogo Fire model [Wilensky, 1997c], which is an extremely simple model of a forest fire (or possibly any other phenomena modeled by percolation across a square lattice). This model has one parameter density, which controls the initial density of trees in the lattice. Since the fire spreads only from tree to neighboring tree, it is easy to predict that the forest fire will die out quickly on a nearly empty lattice, whereas it will burn every inch of the forest if the lattice is completely full. However, we might be interested in how much the outcome is effected by small changes in the tree density. Thus, we could construct a measure of the change in average amount of forest burned at density D and at density $D - 1$. If we were to search for the maximum absolute value of this measure, we would find a striking phase transition that occurs right around a tree density of 60%, as shown in Figure 3.8.

Scientists may have an intrinsic interest in discovering phase transitions in the model's parameter space, simply for better understanding the modeled phenomena. However, in some modeling contexts, the discovery of these phase transitions can also be very important

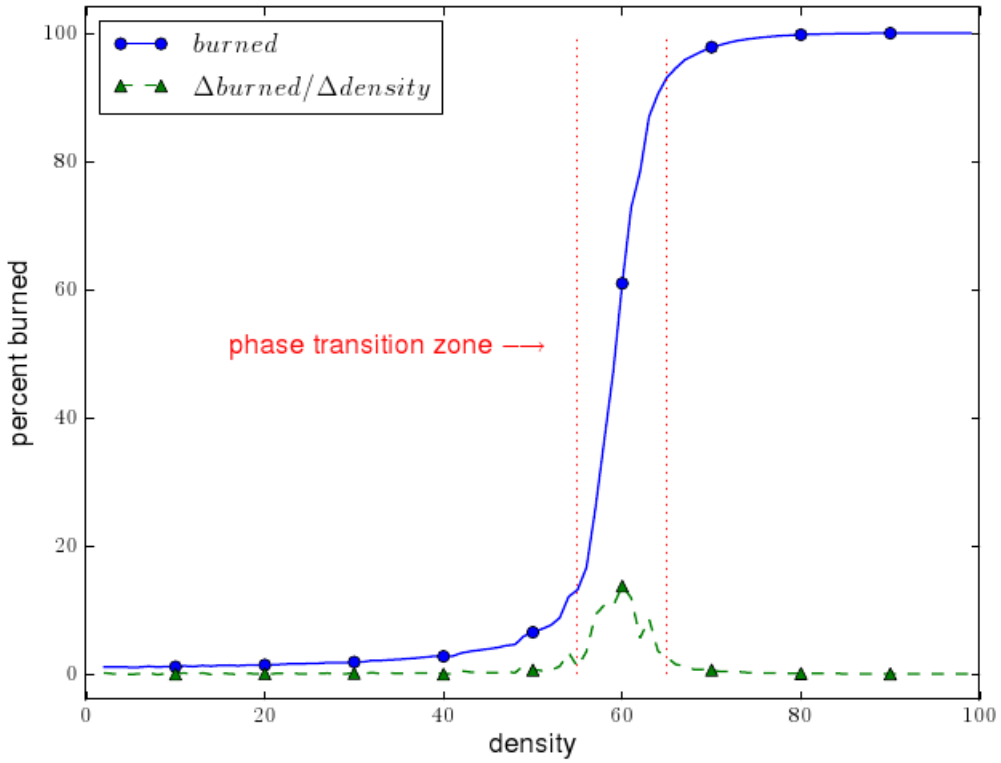


Figure 3.8. Plot showing the amount of forest burned in the NetLogo Fire model [Wilensky, 1997c] as a function of forest density. This plot also shows the relationship between the derivative (as approximated with a unit change in density) and the location of the phase transition around 60% density.

for policy-makers. In some ABMs, model parameters correspond to real-world levers that policy-makers have the power to manipulate, and in these situations phase transitions may represent "leverage points", where a small amount of effort (or money) could go a long way toward effecting change.

3.3.4. Calibration

In Section 3.2.8 above, we discussed how pattern-based measures could be used to compare model results with a *reference pattern* which we might like the model to match. Calibration is the process of changing a model or manipulating its input parameters such that it more closely matches a desired output, or *reference pattern*, which is often derived from empirical/real-world sources, although it could also be the output of a different model. For the purposes of this thesis, we will restrict ourselves to the form of calibration where only the model parameters are being varied and the model's code remains unchanged. Using the QBME framework, to calibrate a model we require an *error measure* (or alternatively a *similarity measure*) which compares the model output to the *reference pattern*. Then genetic algorithms (or other meta-heuristic search algorithms) can be used to find parameters that minimize the error measure (or maximize the similarity measure).

For instance, one might compare the population dynamics of the species in the Wolf Sheep predation model with historical population dynamics in a closed ecosystem being studied by population biologists (as we will do for one of the benchmark tasks in Chapter 9. In this particular case, the reference pattern is a list/vector of numbers corresponding to the historical population values, and the similarity measure uses the correlation coefficients between the real and simulated datasets.

Another example of calibration is explored in Chapter 4, when searching for parameter that cause vee-shaped bird formations. In this case, rather attempting to match specific numeric values, the similarity measure judges how close a formation is to a perfect vee for a range of vee angles obtained from the observation of Canada geese in the wild.

The calibration process is another area where diversity plays an important role: namely, the diversity of results obtained from comparing multiple simulation runs against the empirical data. Variance in these results highlights a trade-off between *how closely* and *how often* the simulation agrees with the reference pattern, an issue that will be discussed in the context of a real calibration problem (the well-known *Artificial Ansazi model*) in Chapter 6. Because calibration is a particularly common and important task for modelers, this topic receives more extensive treatment in the case studies given in both Chapters 6 and 7.

3.3.5. Sensitivity Analysis

Whereas *calibration* is the process of seeing how well a model is able to match its target, under certain conditions, *sensitivity analysis* is concerned with how sensitive the model's results are to changes in its rules, assumptions, or parameters. Again, in the present document, we will restrict ourselves to studying sensitivity analysis with respect to changes in model parameters. Sensitivity analysis is an extremely important (yet sadly often neglected) aspect of agent-based modeling because it provides a basis to evaluate the robustness or fragility of the model. Conclusions drawn from models that are fragile (highly sensitive to specific parameter settings) are not as trustworthy than those drawn from robust models.

Like calibration, sensitivity analysis also relies on some error measure to measure how far off the model result is from the desired reference pattern. As a result, to perform sensitivity analysis in the QBME framework, we may simply search for parameters (within the range that we would like to see that the model is robust) that maximize this error measure, rather than minimize it. If the search is able to find parameter settings that yield a large error measure, this indicates fragility, and also provides information about which parameters the

model is most sensitive to. Sensitivity analysis will be discussed in greater detail in the Artificial Anasazi case study in Chapter 6.

3.3.6. Model Testing

Another essential task in the ABM development process is *model verification*, which is the process of determining whether the ABM is a correct implementation of the model author's conceptual model [Wilensky & Rand, 2007, in press; G. Gilbert, 2008]. More informally, it is checking whether the rules of the model really are what you think they are. Model testing is a form of software testing that can aid in the verification process. Model testing may involve either static analysis of the model's code, or dynamic testing during model simulation. Within the latter category, there are a variety of approaches (such as unit testing, which checks whether individual subcomponents of the model are working correctly), or checking model invariants (e.g., that the total amount of money/energy in the simulated economy/chemical reaction should always remain constant). In many cases, these approaches rely on testing the model with numerous random parameters (or perhaps focused on the boundaries of valid parameter ranges) and checking that certain conditions have not been violated. These are all good approaches to model testing, and should be considered as useful tools in the model verification process, along with non-automated human-based efforts such as code reviews, or model replication attempts [Wilensky & Rand, 2007].

However, building on previous work in using evolutionary algorithms for software/model testing [Miller, 1998; Wakeland et al., 2005], the QBME framework suggests another approach to complement these techniques. Because ABMs have stochastic elements, expecting the model to produce an exact result is often too strict a criterion. However, model authors should be able to offer definite ideas about what range of results the model should be able

to produce (either based on their modeling assumptions or based on analogy to the target phenomena being modeled) . For instance, model authors might believe that the number of wolves in their modeled world will never exceed 1000. A traditional model-checking approach might run the model thousands of times with different random parameter settings, and check if the number of wolves ever exceeds 1000. The QBME approach is similar, but differs in one key aspect. We construct a measure of the maximum number of wolves at any time step, and take the maximum value of that measure across replicate model runs, and then *use a GA* to search for parameters that maximize this measure. The advantage here is that while random sampling of the space may be extremely unlikely to find parameter settings that result in unreasonably large wolf populations, using the maximum number of wolves as a fitness function may efficiently drive the genetic algorithm toward high-wolf parameters. Model testing can also be a beneficial side product of other forms of QBME exploration – for instance, while performing sensitivity analysis in Chapter 6, we discovered a bug in the published Artificial Anasazi model, due to a discrepancy between our intuitions about which model parameters the GA had identified that the model was sensitive to. We also examine one case of explicit model testing in the benchmarks in Chapter 9.

3.4. Applying Measures in Search-Based Exploration

3.4.1. Genetic Algorithms Overview

Before discussing the interaction between behavioral measures and the search algorithms that use them to drive the exploration process, a brief review of genetic algorithms (GAs) in this context may prove helpful. The genetic algorithm starts with a population of individuals. However, when we say "individuals" here we are not talking about individual agents

within the ABM, but rather each individual corresponds to one configuration of the parameter settings for the ABM. The initial population (generation 0) is composed of individuals that represent randomly chosen parameter settings in the space of possible parameter settings being searched. The following steps are repeated. First, the "fitness" of each individual in the population is evaluated according to the behavioral "objective function" (or "fitness function" in genetic algorithms parlance). As mentioned in the sections above on designing behavioral measures, evaluating the fitness function tends to require running the agent-based simulation multiple times with the parameter settings represented by the individual, and it results in assigning the individual a numeric fitness score. Then certain individuals from the current generation are selected for "reproduction". There are various mechanisms (roulette selection, tournament selection, rank-selection, etc.) for selecting these individuals, but they all have the common trait that individuals that have better fitness are more likely to be selected, but that there is some random element to the selection process, so that less fit individuals do have some chance of passing on their genetic material as well. A selected individual (parent) may either undergo sexual reproduction with another parent (using the crossover/recombination operator) to produce two children, or simply undergo asexual reproduction (cloning) to produce a single child. Crossover may be as simple as inheriting certain parameter settings from one parent, and the rest from the other (although it can be slightly more complicated if parameter settings are encoded using lower-level representations such as binary strings). The same individual may be selected as a parent multiple times, and there are no constraints such as monogamous partnerships, etc. The mutation operator is applied to all the "children", though the mutation operator works probabilistically such that mutations may occur (multiple times) in some children and not at all in others. Thus,

some children may end up being identical copies of their parents, while others may be combinations of multiple parents, and some may have been altered in random ways. The number of children created in this manner is equal to the original population size, and it is these children that form the "next generation" of the genetic algorithm[12] The fitness of each child must now be evaluated, and these children will have children, and the cycle will continue. The cycle may stop either when some target fitness level has been reached (corresponding to a configuration of parameter settings that sufficiently elicits the desired behavioral pattern), or when some specified number of generations (or model evaluations) has passed. This process is illustrated in Figure 3.9.

3.4.2. Behavioral measures as fitness functions

Not all behavioral measures are created equal, and this is especially true for the purposes of driving evolutionary search processes. In the context of the Wolf Sheep Predation model, let us suppose that we would like to search for parameters that lead to the extinction of the wolf population. Let us consider two different behavioral measures to serve as fitness functions for this exploration task. We define measure f_A to be the number of wolves after some specified number of ticks (T), and ask the search algorithm to find parameters that minimize this function. We define measure f_B to be 0 (extinct) if there are no wolves left, or 1 (alive) if there are any wolves remaining at time T, and ask the search algorithm to find parameters that give us a result of 0. Both of these measures quantify the "wolf extinction" behavior for us. Furthermore, one could argue that measure f_B marks the extinction criteria more clearly/precisely than f_A. However, in the QBME context, f_B is a poor choice, and

[12]This describes the classic *generational* population replacement strategy, but there are other variants. e.g., In a *steady-state GA*, only a single individual in the population is replaced at a time.

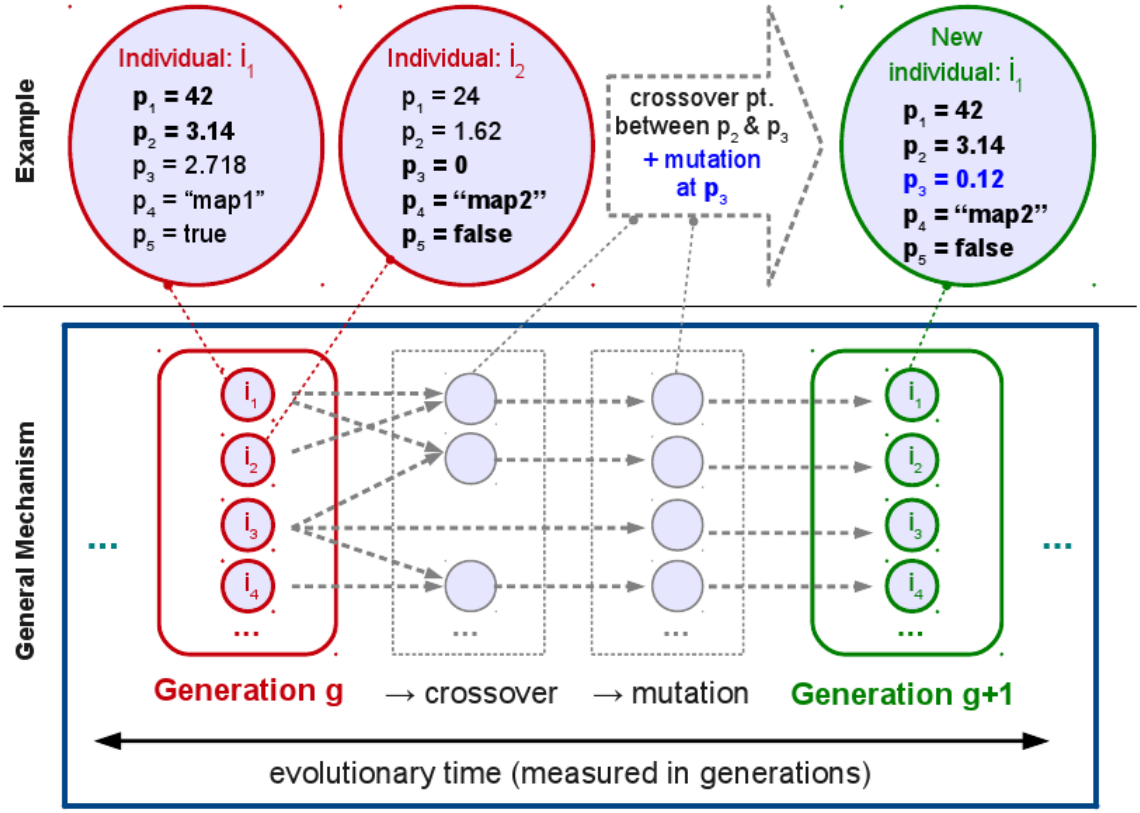


Figure 3.9. Diagram illustrating how a basic genetic algorithm (GA) operates in the context of evolving parameter settings for an agent-based model. Each individual i_j represents one configuration of parameter settings.

f_A is by far the more appropriate measure to use. Why is this? The fitness function must not only identify the desired behavior we are searching for, but it must also help to guide the search process toward its goal. In the case of f_A, the algorithm would have little choice but to keep choosing new parameters randomly until it happened to find a setting which caused the wolf population to reach exactly 0; this is akin to looking for a needle in the proverbial haystack. Measure f_B on the other hand, provides additional feedback to the search algorithm; a relatively low number of wolves suggests a more promising region of the search space to investigate, whereas parameters that yield an extremely high number of

wolves suggest a region to avoid. The fitness function needs to provide a "search gradient", which assists the search algorithm in moving toward local (or potentially global) optima in the search space. A good fitness function provides a measure of how close a set of parameters is to achieving the target behavior (in this case, extinction). One must choose a measure that both provides a wide range of values (so it can serve to usefully discriminate between the goodness of different parameter settings) while at the same time is strongly correlated with the actual behavior which one is trying to elicit. Specifically, for the f_A ("small final population is better") fitness function to be useful for improving the efficiency of the search for extinction, it must be the case that small-population-outcome parameter settings are more likely to be near (in the parameter space) to extinction-outcome parameter settings than large-population-outcome parameter settings are. This seems intuitive, and in fact an examination of a small subspace of the fitness landscapes for these two measures (shown in Figure 3.10 confirms that this is so.

As an additional point about these measures, in order to capture the behavior of "eventually the wolves go extinct", one would wish to choose the time limit T large enough to be confident that the wolf population has stabilized and will continue to thrive or will have gone extinct before that time. However, there are (at least theoretically) conditions where even after millions of time steps of stable population dynamics, a series of chance events could cause the wolves to go extinct. And choose T relatively small, but sufficiently large enough that one posits that the wolves might go extinct under some conditions. One might also wish to search for parameters which cause the wolves to go extinct *only* after a certain amount of time T_{limit} has passed. More subtle behaviors like this are more challenging to quantify, but not insurmountable. In this case the fitness function must reward wolf survival up to a certain time, but then reward extinction after that point. The primary difficulty

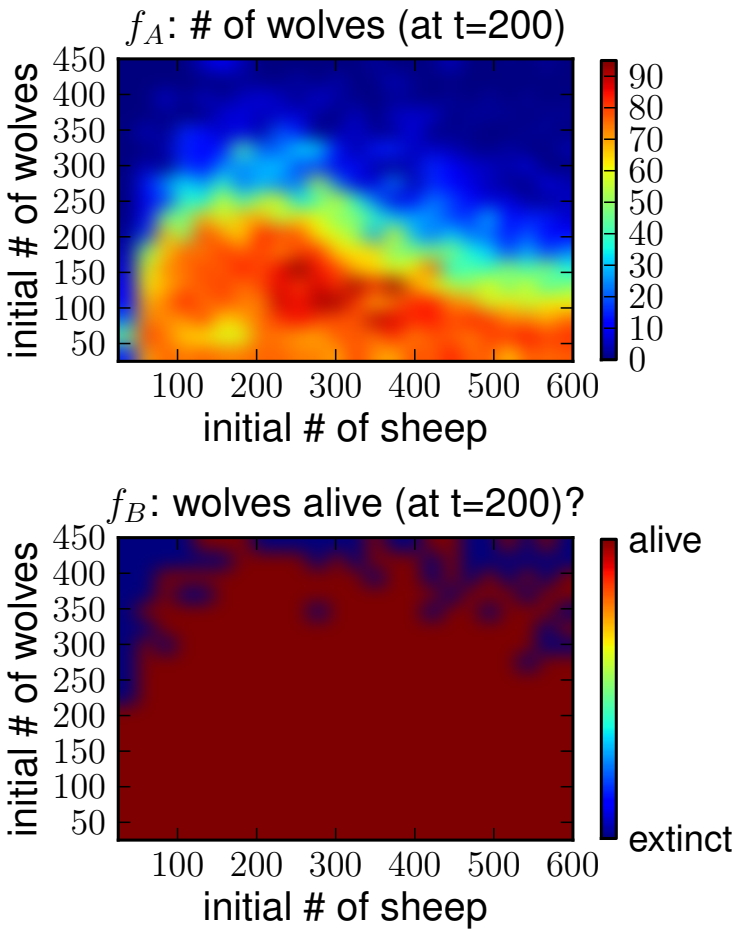


Figure 3.10. These heatmaps show a two-dimensional slice of the eight-dimensional fitness landscape for the Wolf Sheep Predation model under two different fitness functions to search for the extinction of the wolf species. The upper plot fitness function uses the number of wolves remaining, which provides a reasonable gradient which can lead the genetic algorithm toward the extinction zone. The lower plot fitness function simply measures whether the wolves are extinct or not, and thus provides no information that the search algorithm can exploit.

here is that one must be careful not to let one reward measure outweigh (or conflict with) the other. Minimizing the difference between the average population after $T_{threshold}$ and the average population before $T_{threshold}$ may not suffice. This could lead to behavior where extremely large populations that decline and settle at a lower equilibrium, which may score better on the objective function than situations where a small population eventually dies off. When working with genetic algorithms, one learns fairly quickly that you often "get what you search for." But equally often you find that what you searched for wasn't actually what you were trying to find.

It turns out we can avoid the problem in this specific case by combining sub-measures for two incommensurate quantities. One sub-measure ($SM1$) could be the number of steps until wolf extinction, or 0 if the wolves never died out. If we find parameter settings that cause the wolves to go extinct, this sub-measure should help drive the search toward parameters that take longer to go extinct. However, this sub-measure provides no *fitness gradient* for finding wolf extinction in the first place, so we would like to combine this with a measure similar to f_A above, which searches for minimal wolf population, and thus may help us find extinction. But we must be careful not to search for minimal wolf population too soon in the run, or this sub-measure will be in conflict with $SM1$. So for $SM2$ we should choose a time later than $T_{threshold}$ to measure the wolf population. Notice that we wish to minimize $SM2$ while we wish to maximize $SM1$. Thus, for our combined behavioral measure we should minimize $SM2 - SM1$ (or equivalently, maximize $SM1 - SM2$). When combining sub-measures that compare apples to oranges (or in this case *time steps* to *population levels*), it is often good practice to normalize the values of these sub-measures to be between 0 and 1. Thus each sub-measure will be given equal weight in the resulting measure. We can easily normalize $SM1$ by dividing by T_{max} (the maximum number of steps the model is run). Normalizing

$SM2$ is not so simple, however, because we do not know how large the wolf population could be at any point in time. Thus, based on prior experience with the model, we may estimate a reasonable maximum population value for use in normalization. Fortunately, perfect normalization is rarely necessary. In fact, in this particular case, because of mutual exclusion in these two sub-measures, normalization of either measure was not necessary (more elaborate justification of this is left as an exercise to the reader).

3.4.3. Outer levels: measures of the search process

There are several additional levels in which diversity and similarity play important roles in the evolutionary search/exploration process. However, at this point the discussion has moved beyond the level of specifying the single objective function for quantifying behavior to be searched for. Thus, the following is no longer relevant to the creation of quantitative measures, but instead focuses on how the organizing principle of diversity is important within the search process itself, as well as in understanding and interpreting search results.

3.4.4. Diversity in the GA populations

As discussed above, in population-based search mechanisms, such as genetic algorithms, the algorithm maintains a population, or pool of individuals. The population allows the genetic algorithm to take a multi-pronged search strategy, exploring multiple directions in the search space simultaneously. Thus, the diversity of individuals in the population is important to the algorithms success. The mutation operator in particular contributes to the diversity of the population, as higher mutation rates lead to more variation among individuals in the population. Selective pressure is a force that counteracts diversity. Some individual have better "fitness" than others, and are thus more likely to be selected for reproduction.

The more biased the search algorithm is toward the reproduction of fitter individuals, the greater the "selective pressure". As a result certain individuals become overrepresented in the population, and strong selective pressure can lead to the population to converge. The crossover (or recombination) operator can actually contribute to both diversity and homogeneity within the search. Over short time periods, crossover can produce variation by creating novel combinations of genes from individuals in the existing population. However, over long time periods, even in the absence of selective pressure, repeated crossover without mutation will eventually lead the population to converge. However, the population also implicitly acts as a measure of how promising a certain area of the search space is; better areas are more likely to be better represented within the population. Thus, the genetic search process must maintain a balance between population diversity and selective pressure. Diversity promotes broader *exploration* of the search space, by exploring more regions simultaneously, even if they may appear relatively unpromising at first. Selection promotes *exploitation* of the currently most promising regions of the search space, by concentrating population in those regions and thus investigating those regions more thoroughly (and fine-tuning good solutions into better ones). Beyond the standard mutation and crossover operators, some have also proposed explicit mechanisms (such as niching [Mahfoud, 1995; D. E. Goldberg, 1989], Random Immigrants [Grefenstette, 1992b], Triggered Hypermutation [Cobb, 1990], and others [Ursem, 2002; Muhlenbein, 1991]) to help maintain diversity in GA populations, and thus prevent premature convergence on suboptimal solutions. Such techniques may prove useful in the QBME context, but this thesis does not explore their use, which would add an unnecessary level of complication to the basic process. However, incorporating additional diversity maintenance mechanisms into the search process would be a relatively straightforward modification that might provide modest gains in search efficiency.

3.4.5. Variation within search results

Moving up yet another level, we can look at diversity in the results from performing multiples searches. (In this section, we are only discussing multiple executions of the same search algorithm, but with different random seeds / initial conditions – we will discuss the use of multiple search algorithms in Section 3.4.6 below.) First of all, it is important to emphasize that practitioners of the QBME framework should not be satisfied with running only a single search for a desired target behavior. Multiple searches should always be performed. Even though genetic algorithms are often lauded for their ability to escape local optima in the search space, it is quite possible that for any particular search, the population may converge prematurely and stagnate in a poor area of the space. Even if a single search finds parameters that yield great performance on the behavioral measure, it's possible that additional searches will find substantively different parameters that have equally good performance, or perhaps even parameters that are significantly better. Recall that our core mission is the exploration of model behavior, and although we are using optimization techniques to do this, sometimes much can be learned by looking at more than just the single most "optimal" parameter settings discovered using this process. The diversity that may occur among search results can be broken into several distinct cases.

Case 1: The parameter settings found by the searches vary from one another, and the associated behavioral scores also vary considerably. This is a likely indicator that the search process was not successful, particularly if the parameter settings are fairly spread out in the search space, and behavioral scores are mostly poor. There are many possible causes for a failed search, including: not enough time was allowed for

the search, there was too much noise/uncertainty in the fitness function, the trade-off between exploration and exploitation was too far imbalanced (e.g., too high a mutation rate, too small a population size, too strong of selective pressure, etc). If the parameters are not generally spread out, and instead form a few clusters, this probably indicate several local optima in the search space, which the search process was often trapped by. Although there is no general way to guarantee escape from local optima, sometimes they can by avoided by increasing the emphasis on exploration, to prevent premature convergence of the search algorithm's population. On the other hand, while classic optimization researchers view local optima as a bane to be avoided, discovering the locations of local optima in the parameter-space of an agent-based model may actually be useful/informative. Modelers may wish to investigate why, in each local optima, did the combination of parameter settings lead to a higher behavioral measure than the nearby parameter settings.

Case 2: The parameter settings found by different searches can be different, but they all yield similar scores on the behavioral measure. Similar to case (1) above, if there is clustering of the parameter settings, this could indicate several local optima with similar fitness levels. Also as above, modelers would wish to know about these multiple optima, and explore the model behavior more thoroughly in each identified region of the space. If the parameter settings are spread out, this may indicate a large "plateau" in the fitness landscape. That is, rather than a single point in the search space that yields high fitness, there could be a range of settings for the parameters which each elicit the target behavior equally well. For instance, changes in certain parameters may not impact the behavior. In this case, looking at the distribution of search results can be helpful for identifying these "robust" regions of

the parameter space, as well as identifying which model parameter the behavior is more or less sensitive to.

Case 3: The parameter settings vary only a little, but the behavioral scores are substantially different. This is not a particularly common scenario, but may stem from a number of possibilities. A) It may indicate that in the "good" region of the parameter space the fitness function is quite noisy, and running additional replicates of the model in this region is necessary to get a more accurate estimation of the behavioral measure. (It may also indicate that the method of averaging for the fitness function was poorly chosen, with significant outliers affecting the measurement.) B) It may indicate that the searches were consistently able to find the same "good" region of the parameter space, but that in some cases the searches were unable to fine-tune their search results to reach the even better fitness levels that some of the searches happened to achieve. This could be symptomatic of too high a mutation rate, or other factors that encourage exploration too highly, at the expense of exploitation/fine-tuning.

In each of these cases, it is helpful to examine the diversity among search results on a parameter by parameter basis, as often the settings for certain parameters are relatively invariant among the results, whereas others vary widely. This can aid in the identification of which parameters are actually important for eliciting the target behavior, and other parameters which have negligible impact. In fact, it was this observation the diversity of settings among different model parameters which led to the discovery of a bug in the published Artificial Anasazi model, as discussed in more detail in Chapter 6. Variation among search results will arise in several of the case studies in following chapters, and the meaning of this variation will be interpreted more specifically in those contexts.

3.4.6. Variation resulting from multiple search methods

While running multiple repetitions of the same search algorithm is a good idea, for additional confidence that a good solution has not been missed, it may be wise to try several different search algorithms (or different parameterizations of the same search algorithm) when exploring a model's behavior. For a specific algorithm, search parameters (such as population size, mutation-rate, crossover-rate, etc) can be varied. Within the family of genetic and evolutionary algorithms many variants exist, including varying choices for genetic operators, selection methods, and population replacement mechanisms. One could also compare search results with other (non-genetic) meta-heuristic search algorithms such as hill climbing, simulated annealing, particle swarm optimization, Tabu search, or others.

Thus, moving up yet another level, let us consider the situation of observing different results when employing different search methods. First of all, this could arise for the same reasons as variation occurring among search results using the same method (described in the previous subsection). However, if using search method A consistently finds results in a certain region of the parameter-space, whereas search method B consistently finds results in another region, when both search methods are using the same objective function, then additional explanation is necessary. In this case, it could indicate biases among the search methods.

For instance, using the same objective function but with fewer sampling replicates may result in a noisier fitness signal, which could cause a search algorithm to prefer noisier regions of the space (which can occasionally give very high fitness values, although the average fitness behavior is poor. One could argue that this is a difference in objective function, rather than search algorithm, but in fact a similar situation arises when one search method uses *fitness*

caching, while another does not. This subject is discussed in much greater detail in Chapter 8, but essentially caching prevents additional re-sampling replications, which might give a better approximation of the fitness in that region.

As a second example, consider two search algorithms that are identical except for the mutation mechanism. Assume a real-valued genes, on an interval between 0.0 and 1.0, and using additive Gaussian mutation. That is, a gene with current value of x will, after mutation, assume a value of $x_{new} = x + N(0, \sigma^2)$ where $N(0, \sigma^2)$ is a value drawn from a normal distribution with mean 0 and variance σ^2. But what if x_{new} is less than 0.0 or greater than 1.0 – this can happen regardless of the variance σ^2, since the normally distributed random variable can assume arbitrarily high or low values (albeit with small probability)? Some fix is required, to constrain x_{new} in the desired range. Here are three possibilities:

(1) clipping: if $x_{new} < 0$, set x_{fixed} to 0 (and if $x_{new} > 1$ set x_{fixed} to 1).

(2) mirroring: take whatever amount x_{new} is out of range, and move it that far from the boundary in the opposite direction. e.g., if $x_{new} = -0.3$, x_{fixed} would be 0.3, and if $x_{new} = 3.1$, then x_{fixed} would be 0.9 (after mirroring multiple times).

(3) rejection-based sampling: if x_{new} is out of range, regenerate $x_{new} = x + N(0, \sigma^2)$ until x_{new} is in range.

Note that the first of these has a fairly strong bias toward genes taking on the extrema values of 0 and 1, whereas the other two methods do not. Although the word "bias" often has negative connotations, in some cases this bias could indeed be beneficial for the search process, since the extrema of a parameters range are commonly (though not always!) settings that cause the most or least of a certain behavior to occur. However, the point is that search mechanisms using different mutation operations may lead the search to sample certain areas

of the parameter space more often than other areas, and this could lead to differing results from different search mechanisms.

These were just two illustrative examples, but in fact the situation is more widespread. It turns out that *all* search methods are in some way biased in their approach to sampling the search space, with the exception of random search, which samples uniformly at random. Again, the word "bias" must not be perceived as negative here – in fact, all intelligent search techniques depend on these sampling biases, which take advantage of structure in the fitness landscape in order to outperform random sampling of the space. Thus, there is always the potential for two search methods to be lead down paths toward different regions of the search space, and thus tend toward different results. In such cases, the region of the parameter space that give superior fitness values is generally preferable, but the regions identified by other search mechanisms may deserve examination as well. However, in practice, a large variety of search techniques are likely converge to the same high-performing region of the search space, if given enough time, and appropriate search parameters.

3.4.7. Variation of search results from differing specifications

There is one final method of exploration in the QBME framework that we haven't touched on yet. That is, by comparing the results from performing searches with varying specifications, we have the potential to learn many interesting things about the model, or about differences between models. This approach can be broken down into three cases: searching different parameter ranges, and searching different models, and searching for different objective functions.

Searching different parameter space ranges. One approach is it to search for the same objective function, but in different subspaces of the full parameter space. By constraining

the search process to a certain range of parameters, one can discover the parameters *within that range* that maximize the target model behavior. By changing to a different range of parameters, and searching again, one can compare a) the parameter settings found, and b) the extent to which the behavior was manifested in each region. This is probably best illustrated by a couple of examples.

In the Wolf Sheep Predation model, a parameter called grass? determines whether the grass (food source for the sheep) is limited in supply and regrows over time, or whether the grass is assumed to be unlimited. Including grass growth/consumption in the model adds another trophic level to the model's food chain, and generally produces much more stable population dynamics than in the purely two-species variant. However, one may wonder whether there are choices for the other parameters of the model such that the inclusion of grass would cause the population dynamics to be even less stable than in the other case (perhaps using a measure of the frequency of wolf/sheep extinction). To examine this, the parameter space may be split into two subspaces: one where the grass is limited, and one where the grass is unlimited. All other model parameters (initial-number-sheep, initial-number-wolves, etc) are unconstrained. Thus, two separate searches for population instability can be run – one in each subspace – and the result will be the most "instability-causing" parameters in each case. These parameters can then be compared, both to see which scenario permitted greater instability, and also to see whether similar parameter settings contributed to this in both cases. As a second example, in the NetLogo Fireflies model there are two general strategies ("advance" and "delay") which the fireflies can use to attempt to synchronize with one another. One could search the parameter space for each of these cases, to determine which of these strategies permits the quickest convergence to synchronization under varying other conditions (cycle length, length of flashes, etc). This same idea is applied in Chapter

5 in the real-world setting of examining the effects of different social network topologies on viral marketing campaigns. In this case, the different subspaces being examined correspond to the different social network structures that the model can be initialized with.

Searching different models. Alternatively, one can apply the same objective function to explore two (or more) distinct agent-based models. For instance, suppose that two different agent-based models are proposed by different teams of researchers to explain a certain phenomenon, such as the boom/bust cycle of the stock market. To explore the range of behavior these models are capable of, one might construct a measure of market volatility that can be applied to each model. Searching for the parameters that give the most (or least) volatile behavior can help us compare the models strengths and weaknesses. Using a calibration measure, we can search for parameters that cause each model to best align with an empirical dataset. On the other hand, we can perform sensitivity analysis on each model separately, and thus compare how sensitive/robust each of the models is to changes in its parameters. Even though each of the models may have a different set parameters, by construction behavioral measures that can equally be applied to different models of the same phenomenon, we can use QBME methodology to explore those models in tandem. We will see an example of this in the case study in Chapter 4, which compares two models of collective animal movement to see the extent to which each is capable of producing a certain behavior.

Searching for different objective functions. This approach is one of the most obvious extensions of the previous discussion. While finding parameters that yield a certain behavior is informative, the development and maximization of *single* behavioral measures really only scratches the surface of the query-based model exploration paradigm's potential. It's possible to search (sequentially or in parallel) for a variety of behaviors of interest, and thus form

a larger mental map of the parameter space. Similar-seeming behaviors which you might believe to be well-aligned may result from qualitatively different parameter settings. In particular, it's useful to look at the degree to which behavior-maximizing parameters coincide in the space. For instance, does searching for parameters that yield greatest wolf longevity also yield the highest average wolf populations? Intuitively one might feel these are connected, but one's intuitions about ABM behavior can often be wrong. On the other hand, one might discover that similar parameter settings maximize two seemingly unrelated behaviors, raising new and unexpected questions. By performing searches for different behaviors in turn, and then comparing the results, one can both refine existing theories and form new hypotheses to explain how the model's parameters affect model behavior.

3.4.8. Iterative exploration

In order to integrate the QBME framework with more typical exploratory model analysis, a few words about applying it in practice will be helpful. The most important thing is that model exploration is an *iterative process* that builds on itself. One doesn't start with a single question, design a measure to attempt to answer that question, run a search, draw a conclusion, and then be "done" exploring their model. The exploration process is much longer, richer, and messier than that. Modelers usually start with a variety of questions in their mind, though they may focus on one or two at first. However, as they learn more about model behavior, they continually refine the questions they are asking about the model, and the results of each search will provide insight into certain questions, while opening up new questions about model behavior. And oftentimes the questions are not precisely formed: "Does the model ever do something crazy?", "Why doesn't it settle down to an equilibrium more often?". As mentioned above, QBME was never intended to replace other methods of

exploration - rather it complements them by providing a way to answer certain types of modeler questions that were very difficult to answer previously. Not all questions are amenable to the QBME approach. Sometimes running the model with a few carefully human-chosen parameters can answer a question much more effectively than a lengthy automated genetic search. However, you can often gain at least partial insight into even the hardest or most nebulous questions through an appropriate (and sometimes highly creative) transformation of the question into a type that QBME can answer. At present, this transformative question re-phrasal process is more of an art than a science. However, I do not feel that "sciencifying" this process is the most pressing concern – that will come with time and experience. Rather, I believe the greatest hurdle to integrating QBME in practice is is simply to get people shift their exploratory paradigm, so that it *occurs to them to try* rephrasing their questions into queries that automated search-based exploration could shed light on. Simultaneously, new low-threshold tools (see Chapter 10) are needed to make the process easy enough to learn and convenient enough to use in practice.

CHAPTER 4

Case Study 1: Flocking/Swarming Behavior

"Birds of a feather flock together."

– Ancient Proverb

"You can't think about thinking without thinking about thinking about something."

– Seymour Papert

I have always loved Seymour Papert's quote about "thinking about thinking", and I *think* it can be usefully adapted to many other situations, if broadly construed. Specifically, you can't think about exploring agent-based models in general, without thinking about exploring some specific agent-based model. Hence the necessity of case studies. This chapter provides a first case study of using genetic algorithms to explore behavior in two agent-based models of flocking/swarming behavior (now frequently cataloged under the more general moniker of "collective animal motion" models). As described in Chapter 1, the case study chapters have been written to stand on their own, and thus a small amount of repetition regarding motivation for the work and background literature is to be expected, and certain elements of the QBME framework are restated in this context. This chapter demonstrates the use of computer-aided model exploration, showing how evolutionary search algorithms can be used to probe for several qualitative behaviors (convergence, non-convergence, volatility, and the formation of vee shapes) in two different flocking models. This is accomplished by using

BehaviorSearch, the new software tool I created for performing parameter search on ABMs created in the NetLogo modeling environment. Of particular note in this chapter is the importance of recognizing and interpreting the variance in parameter settings, as well as the use of exploratory methods to compare across models. The results regarding the performance of the genetic algorithm relative to other search algorithms are less decisive – a matter that will be returned to in Chapter 9.

4.1. Motivation

Agent-based modeling is a powerful simulation technique in which many agents interact according to simple rules resulting in the emergence of complex aggregate-level behavior. This technique is becoming increasingly popular in a wide range of scientific endeavors due to the power it has to simulate many different natural and artificial processes [S. Bankes, 2002; Bryson et al., 2007; North & Macal, 2007; Wilensky, 2001]. A crucial step in the modeling process is an analysis of how the system's behavior is affected by the various model parameters. However, the number of controlling parameters and range of parameter values in an agent-based model (ABM) is often large, the computation required to run a model is often significant, and agent-based models are typically stochastic in nature, meaning that multiple trials must be performed to assess the model's behavior. These factors combine to make a full brute-force exploration of the parameter space infeasible. Researchers respond to this difficulty in a variety of ways. One common approach is to run factorial-design experiments that either explore model behavior only in a small subspace or explore the full space but with very low resolution (which may skip over areas of interest). A second common approach is to vary only a single parameter at a time, while holding the other parameters constant, and observe the effect of changing each parameter individually. However, because

ABMs often constitute complex systems with non-linear interactions, these methods risk overlooking parameter settings that would yield interesting or unexpected behavior from the model.

As an alternative, we argue that many useful model exploration tasks may instead be productively formulated as *search problems* by designing appropriate objective functions, as we will demonstrate by example in the domain of simulated flocking behavior. We also introduce a new software tool (*BehaviorSearch*), which we have created for the purpose of searching/exploring ABM parameter spaces. Using *BehaviorSearch*, we offer a case study showing how search-based exploration can be used to gain insight into the behavior of two ABMs of flocking that have been implemented in the NetLogo modeling environment [Wilensky, 1999; Tisue & Wilensky, 2004]. We also provide a comparison of the performance of three different search algorithms on several exploratory tasks for these two ABMs. In particular, we will show how genetic algorithms and hill-climbing can be used to discover parameter settings for these models that yield behaviors such as convergence, non-convergence, volatility, and specific flock shape formation. This approach can be useful for researchers to better understand the models they have created, the range of behavior their models are capable of producing, and which parameters have large impact on which behaviors. Flocking behaviors were chosen for this case study because flocking is a well-known example of a successful agent-based model, and can demonstrate a wide range of behaviors depending on the controlling parameters.

4.2. Related Work

Rather than using a full factorial experiment design for sampling points in the space, several more sophisticated sampling algorithms exist (e.g., Latin hypercube sampling, sphere-packing). These algorithms stem from the design of experiments (DoE) literature or more specifically the more recent design and analysis of computer experiments (DACE) literature (see [Sanchez & Lucas, 2002] for a discussion of applying DACE methodology to ABMs). While appropriate experimental designs provide efficient sampling of the space in some situations, this is a separate direction from the search-oriented approach that we are pursuing here. In particular, we are interested in the use of genetic algorithms [J. Holland, 1975] (GAs) to search the ABM parameter spaces for behaviors of interest. Genetic algorithms have proven to be quite successful on a wide range of combinatorial search and optimization problems, and are thus a natural meta-heuristic search technique for this task. There is prior work on parameter-search and exploration in ABM, and considerably more on the problem of parameter-search in general.

Calvez and Hutzler [2005] have previously used a genetic algorithm (GA) to tune parameters of an ant foraging model, and discuss some of the relevant issues for applying GAs to ABM parameter search. However, in this case, the GA's performance was not compared to any other method, and the effectiveness of GAs for the ABM parameter search task has not been thoroughly investigated. Our present work contributes toward this goal. Specifically, we compare the performance of a genetic algorithm against a stochastic mutation-based hill-climber, as well as uniform random search, to serve as a baseline for comparison. We also explore a different domain (i.e. flocking models rather than ant foraging), and thus provide another perspective on the issue of automated model exploration.

Genetic algorithms have also been used to attempt to calibrate agent-based models with aggregate-level equation-based models as part of the SADDE methodology [Sierra et al., 2004] for designing ABMs. Our current case study places an emphasis on exploration, as opposed to calibration or model design. The modeler may pose a question about the model's behavior which are potentially interesting, and the distribution of search results should answer that question, and may give additional insight into the interaction between parameters as well.

Other methods of exploration (besides genetic algorithms) have previously been considered. Most notably, Brueckner and Parunak [2003] proposed a meta-level multi-agent system to adaptively select points in the parameter-space to evaluate. This swarm-based approach resembles particle swarm optimization [Kennedy et al., 1995] in that it uses a population of agents that combine global and local information to choose a direction to move in the search space, but it also considers whether to run additional simulations to improve the confidence of results at locations in the space. Brueckner and Parunak also mention in passing that genetic algorithms would be an appropriate choice for this type of search problem, but they did not follow this path, and only offer results from the novel multi-agent optimization algorithm they proposed. A comparison of genetic algorithms with this, and other swarm-based approaches, would be an interesting area for future work.

Genetic algorithms have also been employed in parameter-search problems which are not ABM, but closely related fields. For instance, genetic algorithms have been applied to search for rules in cellular automata (CA) that will produce a certain behavior (e.g., density classification) [Mitchell et al., 1996]. Cellular automata models could be considered a highly restricted case of agent-based models, and the cell state transition rules could perhaps be considered the *parameters* of such models, in which case this would constitute searching the

parameter space. However, the density-classification task is arguably closer to a multi-agent system coordination problem, rather than an agent-based model; the goal here is to get a set of artificial agents (cells) to work together to solve some problem, not to simulate some natural phenomena. Also, while CA rules are naturally represented by binary switches, agent-based simulations tend to have a mix of parameter types, with numeric parameters being the most common.

Our present investigation is also inspired by Miller's work on active non-linear testing [Miller, 1998], which demonstrated the use of meta-heuristic optimization (genetic algorithms and hill climbers) for searching the parameter-space of the *World3* simulation, a well-known system dynamics model (SDM). Our work departs from Miller's in two respects: 1) model stochasticity (which is less frequently present in SDMs) is not addressed in those experiments, and 2) the characteristics of search spaces produced by agent-based models likely differ from those which are produced by aggregate equation-based models.

4.3. Methods

4.3.1. Flocking Models Overview

For our case study we explore the parameter-space of two agent-based models, searching for a variety of target behaviors. The two ABMs are the Flocking model [Wilensky, 1998] (denoted as *Flocking*) and the Flocking Vee Formations model [Wilkerson-Jerde, Stonedahl, & Wilensky, 2010] (denoted as *Flocking VF*). While the parameters of these two models are discussed briefly below, an in-depth discussion of these models is beyond the scope of this document. Thus, we invite interested readers to examine the models themselves, which are both available in the NetLogo models library.

Flocking closely resembles the seminal ABM of swarming behavior in artificial birds (playfully dubbed "boids") that was introduced by Reynolds as a way to create life-like cinematic animation of flocking birds or other flying/swimming/swarming creatures [C. W. Reynolds, 1987]. The behavior of each "boid" is influenced by three basic rules, which provide impetus toward alignment, coherence, and separation. The relative influences of each are controlled by the parameters max-align-turn, max-cohere-turn, and max-separate-turn, respectively. Additionally there are parameters controlling the distance at which birds have knowledge of other birds (vision), and the minimum distance of separation which birds attempt to maintain (minimum-separation). For this first model, exploratory search tasks include the discovery of parameters that yield quick directional convergence (Section 4.4.1), non-convergence (Section 4.4.2), and volatility of the aggregate flock's heading over time (Section 4.4.3).

Flocking VF is based loosely on an extension of Reynolds' work that was proposed by Nathan and Barbosa [2008], attempting to produce the vee-shaped patterns often observed in large migratory birds, such as Canada geese. *Flocking VF* has 8 controlling parameters, which account for fine-grained control over bird vision (vision-distance, vision-cone, obstruction-cone), takes into account benefits of "updraft" from nearby birds (updraft-distance, too-close), as well as flying speeds and acceleration (base-speed, speed-change-factor, and max-turn). The final exploratory search task is to seek parameters that best yield V-shaped flock formations, in both *Flocking* and *Flocking VF* (Section 4.4.4).

4.3.2. Search Algorithms

For each search task, we tested three different search algorithms: uniform random search (RS), a random-mutation hill climber (HC), and a genetic algorithm (GA). For all of the search methods, each ABM parameter's value was encoded as a sequence of binary digits

(bit string) using a Gray code[1], and all the parameters' bit strings were concatenated to create a string that represents one point in the parameter-space. A bit string is evaluated by decoding it into the ABM parameter settings, and running the model with those parameters.

The RS method simply generates one random bit string after another, and in the end chooses the one that best elicited the desired model behavior. RS is a naive search technique, which we included as a baseline for comparison, to determine whether using more sophisticated meta-heuristics (such as the HC and GA) were indeed helpful.

Our HC is primarily a local search algorithm. It starts with a random bit string (s). A new string (s_{new}) is generated from s (each bit of s gets flipped with probability 0.05, which is the *mutation-rate*). If s_{new} is better than s (generates behavior that judged closer to the desired target behavior), then the HC chooses s_{new} as the new s, and the process repeats. If the HC becomes stuck (after 1000 unsuccessful move attempts), it will restart at a new random location in the search space, which makes this a quasi-local search method.

Our GA is a standard generational genetic algorithm [J. Holland, 1975], with a population size of 30, a crossover rate of 0.7, and a mutation rate of 0.05, using tournament selection with tournament size 3. The GA is a more sophisticated search mechanism than HC or RS, and there are several reasons to believe that it might perform better. First, the GA is population-based, which allows it to explore multiple regions of the space simultaneously (more of a global search technique). Second, genetic algorithms have previously been shown to perform well on a variety of nonlinear and multi-modal search/optimization problems. Third, genetic algorithms (like the biological processes of evolution that inspired them) often have a way of coming up with creative or unexpected solutions to a problem, which humans

[1]A high-order binary encoding requires flipping 4 bits to change from 7 (0111_2) to 8 (1000_2). In a Gray code, consecutive numbers only require a single bit flip, thus creating a smoother mapping from numbers into binary search spaces [Caruana & Schaffer, 1988; Whitley, 1999].

would not have considered. However, depending on how the search space is structured, simpler approaches may be more effective. For example, it was shown in one case that a HC performed better on a problem that was specifically designed with the expectation that GAs would work well on it [Mitchell, Holland, & Forrest, 1994]. One important consideration is whether there are so-called *building blocks* in the solution-space, which the GA is able to discover and combine (via genetic crossover) to form better solutions. Phrased at the level of the agent-based model, this question becomes: are there certain combinations of several parameter settings, each of which partially produce desired target behavior, and when combined together produce that behavior even more strongly? If so, the GA may be able to take advantage of that structure in the search space to efficiently find solutions. This notion of *building blocks* may appear counter to the earlier argument that GA's do well in multi-modal nonlinear search spaces, but it is not a paradox. One could imagine a perfectly decomposable problem, where the fitness of each individual is exactly the sum of the fitness contributed by each building blocks, and there are no overlapping building blocks; this would result in a unimodal function with certain linearly additive properties. However, it is possible for complex nonlinear functions to also contain building blocks, where the building blocks may overlap and interact with each other in nonlinear ways to form rough multi-modal fitness landscapes. For one class of complex search functions designed specifically to exhibit building block structure, see Holland's [2000] hyperplane defined functions (hdf's).

The objective function (or "fitness function" in the parlance of evolutionary computation) was always averaged across 5 model runs (replicates) with different random seeds, to reduce variability stemming from model stochasticity. While this variability is essentially "noise" from the search algorithm's perspective, it is simply reflecting the fact that running the ABM results in a range of behavior depending on the initial placement of the birds. Our objective

functions are attempting to characterize the presence or absence of a certain behavior *on average*, and short of running the simulation with every possible initial condition (which is impossible), there will always be some uncertainty about the objective function measure. Taking the average value from several replicate runs of the simulation, however, reduces this uncertainty and smooths the search landscape.

The objective functions were different for each task, and will be discussed individually in each of the investigations below (Sections 4.4.1-4.4.4). For efficiency, objective function values were cached after being computed.[2] The search algorithms were stopped after they had run the ABM 12000 times. Each search was repeated 30 times (except for the *volatility* exploration in Section 4.4.3, which was repeated 60 times for improved statistical confidence), to evaluate search performance and ensure that search findings were not anomalous. To perform these searches, we use the software tool *BehaviorSearch* [Stonedahl & Wilensky, 2010a], which will be discussed in greater detail in Chapter 10.

4.4. Explorations

4.4.1. Investigation 1: Convergence

The convergence of swarm-based systems is one potential property of interest, and has been formally studied for some theoretical cases [Cucker & Smale, 2007]. Thus, the first behavior of interest for the *Flocking* model was the ability of birds starting at random locations and headings to converge to be moving in the same direction (i.e. directional, not positional, convergence). In order to make the search process effective, we must provide a quantitative

[2]The goal of caching is to avoid repeating expensive computations. However, because the model is stochastic, re-evaluating points in the search space could lead to different results than the cached values, meaning that the search process is *affected* by caching. For further discussion of noise/uncertainty and fitness caching, see Chapter 8.

measure to capture the rather qualitative notion of convergence. This quantitative measure (the objective function) will provide the search with information about how good one set of parameters is, relative to another, at achieving the goal. Specifically, we would like to find parameters that yield very little variation between birds' headings. Thus, we will attempt to minimize the following objective function:

$$f_{nonconverged} = stdev(\{v_x(b) \mid b \in B\}) + stdev(\{v_y(b) \mid b \in B\}) \tag{4.1}$$

where $v_x(b)$ and $v_y(b)$ are the horizontal and vertical components of the velocity[3] of bird b, and B is the set of all birds. The standard deviation (*stdev*), which is the square root of the variance, serves as a useful measure of the variation for velocity, and we must apply it in both the x and y dimensions. A value of $f_{nonconverged} = 0$ would indicate complete alignment of all birds. We measure $f_{nonconverged}$ after 75 ticks (model time steps). While 75 ticks is effective here for finding parameter settings that cause the flock to quickly converge, if we were instead interested in the long-term behavior of the system, a longer time limit would be more appropriate. In terms of the query-based model exploration (QBME) framework for formulating measures to quantify behavior (see 3.2), we are using intra-agent measures of v_x and v_x that are being combined by an agent group level *stdev* measure (that quantifies the diversity of some property of agents in the group). At the temporal level of analysis, we are discarding behavioral information from the beginning of the run, and only measuring behavior at the time-slice $t = 75$. (For a lengthier discussion of the QBME conceptual framework for measure formulation, refer back to Chapter 3.)

[3]In NetLogo it is usually more natural to think in polar coordinates or "turtle geometry", where each bird's velocity is represented by its speed and *heading* (angle). However, taking the stdev (or mean) of a set of angles is problematic due to the discontinuity between 359 degrees and 1 degree, so we expressly split the velocity into Cartesian components to avoid this issue.

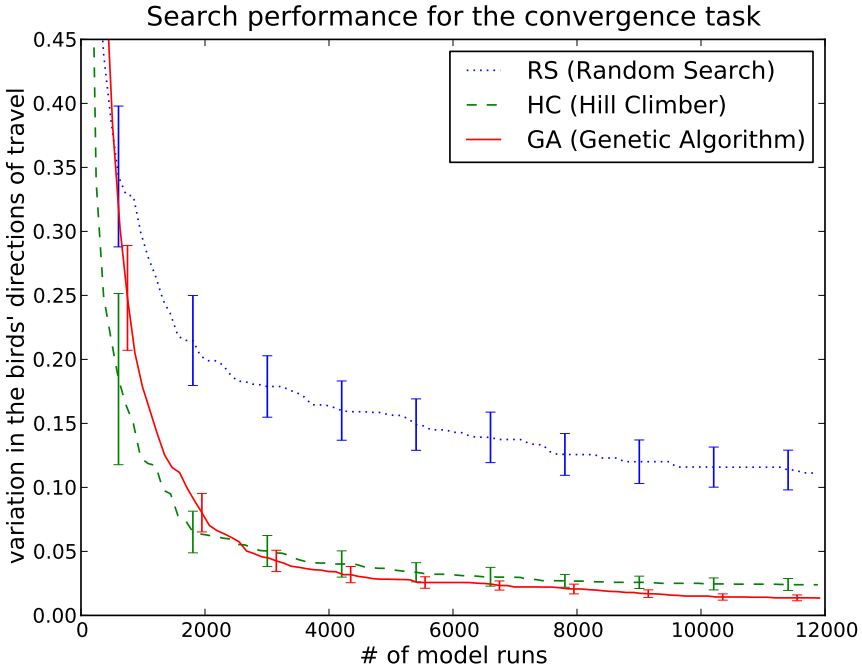


Figure 4.1. Search performance for the convergence task, comparing how efficiently the GA (*genetic algorithm*), HC (*hill climber*), and RS (*random search*) can find parameters that cause the flock to quickly converge to the same heading. (Error bars show 95% confidence intervals on the mean.)

The plot of search progress (Figure 4.1) shows that on average the HC may have found better model parameters early in the search, but in the end the GA's performance was superior (t-test, $p < 0.01$). Both GA and HC significantly outperformed random search. The best parameters found in each run are displayed in Figure 4.2. Examining the resulting parameters is a key step of the QBME process. As modelers we are often interested in the extent to which the model exhibits the behavior we have quantified, but we are often even more curious about what settings of the parameters conjured up such behavior, because this information can lead us to causal or mechanism-based explanations for the behavior. Furthermore, it is preferable to look at the collection of parameters returned by a number of searches, rather than focusing only on the single best parameter setting discovered. As

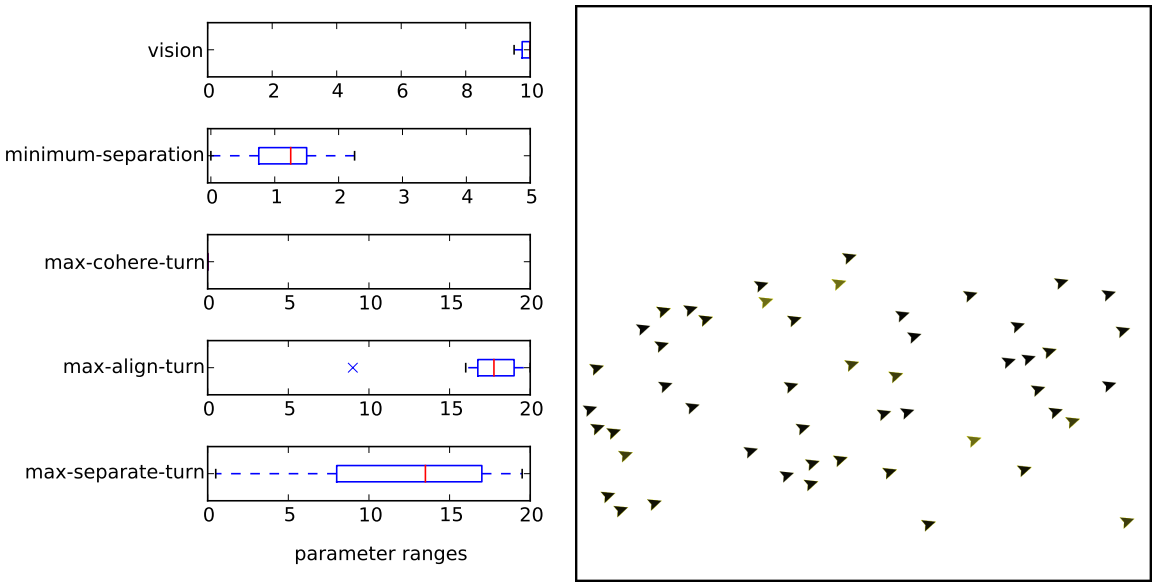


Figure 4.2. *LEFT:* Distribution of model parameter settings found to cause quickest convergence in each of the 30 GA searches. All box-and-whisker plots presented in this chapter show the median line within the lower-to-upper-quartile box, with whiskers encompassing the remainder of the data, apart from outliers which are marked with x's. *RIGHT:* Visualization of the flock (after 75 model steps) using the best parameters the GA discovered.

discussed in the QBME framework (in particular, see Section 3.4.5), diversity among the parameters returned by the searches can provide additional insight into model behavior. In this case, Figure 4.2 shows us that it is crucial for birds to have long-range vision, and that even a small urge to cohere is detrimental to convergence. On the other hand, the wide spread for max-separate-turn suggests that convergence is not very sensitive to this parameter (given the other parameter settings). Often these observations align with our intuitions about the model – for instance, a larger vision naturally allows information to travel more quickly between agents, and thus is beneficial for convergence to a common state.

Figure 4.2 also shows one possible converged state from running the model using the best parameters found by the GA.

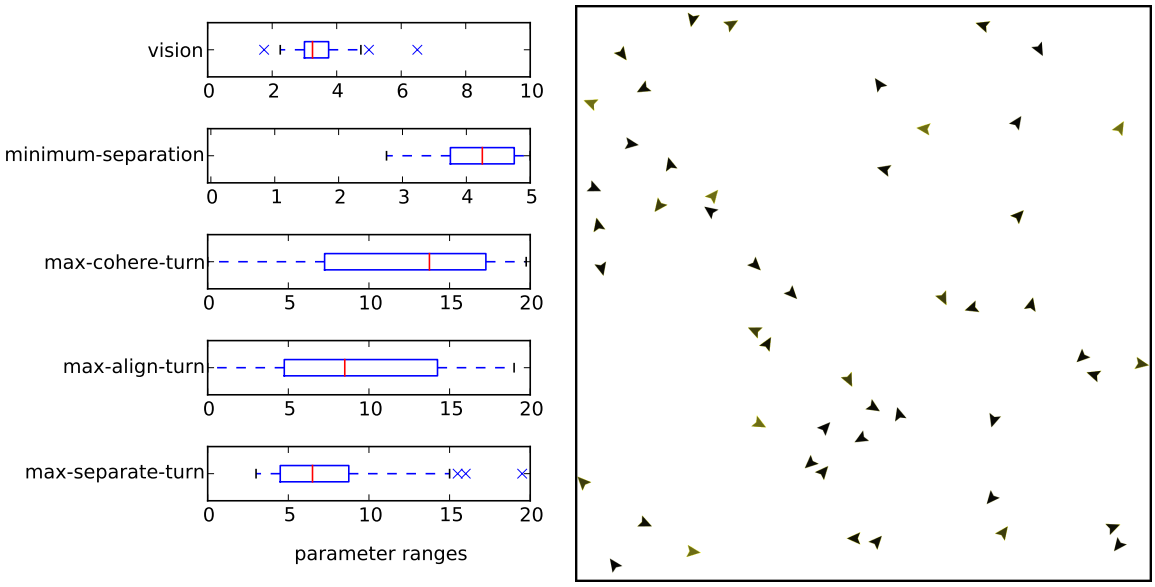


Figure 4.3. *LEFT:* Distribution of model parameter settings found to cause non-convergence in each of the 30 GA searches. *RIGHT:* Visualization of a non-converged flock using the best parameters the GA discovered.

4.4.2. Investigation 2: Non-convergence

Next, we probed for parameter settings that cause the birds not to globally align. For this task, we simply maximized the same objective function we minimized in Section 4.4.1. This task turned out to be rather trivial, as all three search methods (GA, HC, and RS) very quickly found parameter settings that yielded little or no flock alignment. That such behavior is rather common in the parameter space is illustrated by Figure 4.3, which shows a wide distribution of best parameters. The results suggest that for non-convergence, it is helpful for birds to have a low-to-medium vision range, desire a large amount of separation from each other (minimum-separation), and act to achieve the separation (non-zero max-separate-turn). Digging deeper, the results tell us that it is the relationship between parameters that matters; if minimum-separation is larger than vision each bird will seek to avoid any other bird as soon as it sees it, as separation takes precedence over the align/cohere rules.

4.4.3. Investigation 3: Volatility

Our third experiment sought parameters for the *Flocking* model that would yield the most volatility (or changeability) in global flock heading. In contrast to Sections 4.4.2 and 4.4.1 above, we are now interested in looking at model behavior across time, rather than static snapshots of the model state. Volatility is an important, though general, concept in agent-based modeling. In this instance, we are specifically interested in the volatility of an aggregate-level property (group heading) over time. In a model of collective animal motion, the type of volatility we seek relates to coordinated flock/school movement: how quickly can the entire group respond and change from moving in one direction to moving in another? Behavior of this type is important in real-world flocks, swarms, and schools for predator and/or obstacle avoidance. To seek volatile flock behavior, we attempt to maximize $f_{volatility}$, as defined in (4.4).

$$\overline{v_x}(t) = mean(\{v_x(b) \mid b \in B\} \text{ at tick } t \tag{4.2}$$

$$\overline{v_y}(t) = mean(\{v_y(b) \mid b \in B\} \text{ at tick } t \tag{4.3}$$

$$f_{volatility} = stdev(\overline{v_x}(t) \text{ for } t = 400..500) + stdev(\overline{v_y}(t) \; for \; t = 400..500) \tag{4.4}$$

Again, on average the GA was slightly more successful than the HC in eliciting flock heading volatility, and both significantly outperformed random search (Figure 4.4). Only 5 out of the 60 GA searches' best parameter settings had a non-zero value for minimum-separation, indicating that birds flying close together is a key factor for maximal volatility. Long-range vision, and large effects of max-align-turn and max-cohere-turn are also important (see Figure 4.5). The flight pattern of a flock exhibiting considerable volatility is shown

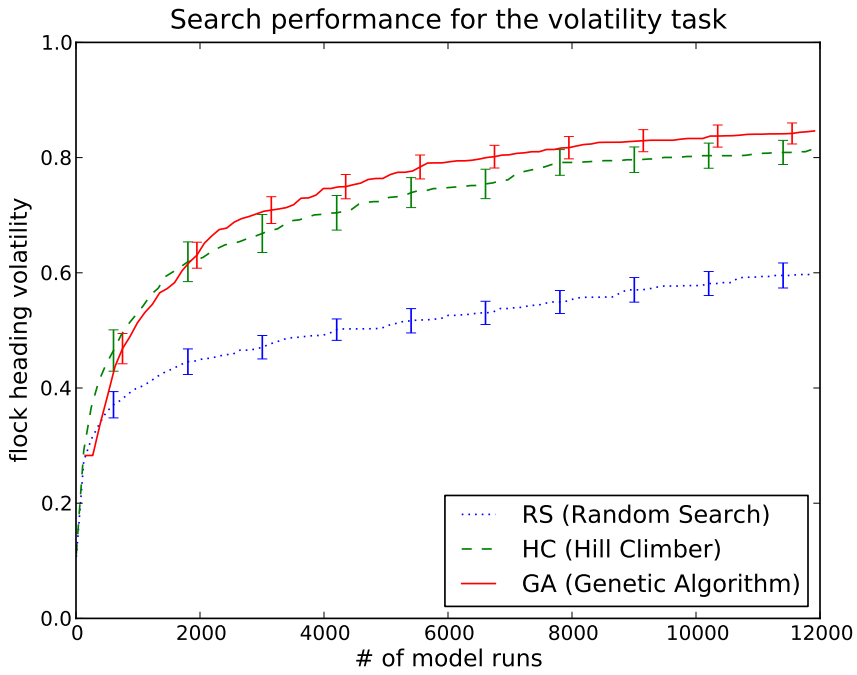


Figure 4.4. Comparison of search algorithm performance for the flock heading volatility task. The final mean performance of the GA was better than the HC (t-test, $p < 0.05$), but not substantially so. (Error bars show 95% confidence intervals on the mean.)

in Figure 4.5. The single bird positioned at the left side in the rear is at least partially responsible for shift in flock heading, because of the strong coherence parameter.

Despite taking the average of 5 replications, noise due to model stochasticity was still significant. For example, the search reported finding settings yielding 0.99 volatility, but averaging 1000 runs at those settings showed true volatility of 0.41. This fact could bias the search toward parameters that occasionally yield very high volatility, over those that consistently yield moderately high volatility. Both goals are potentially interesting for model exploration; however, appropriate noise reduction methodology is a worthy subject for future research.

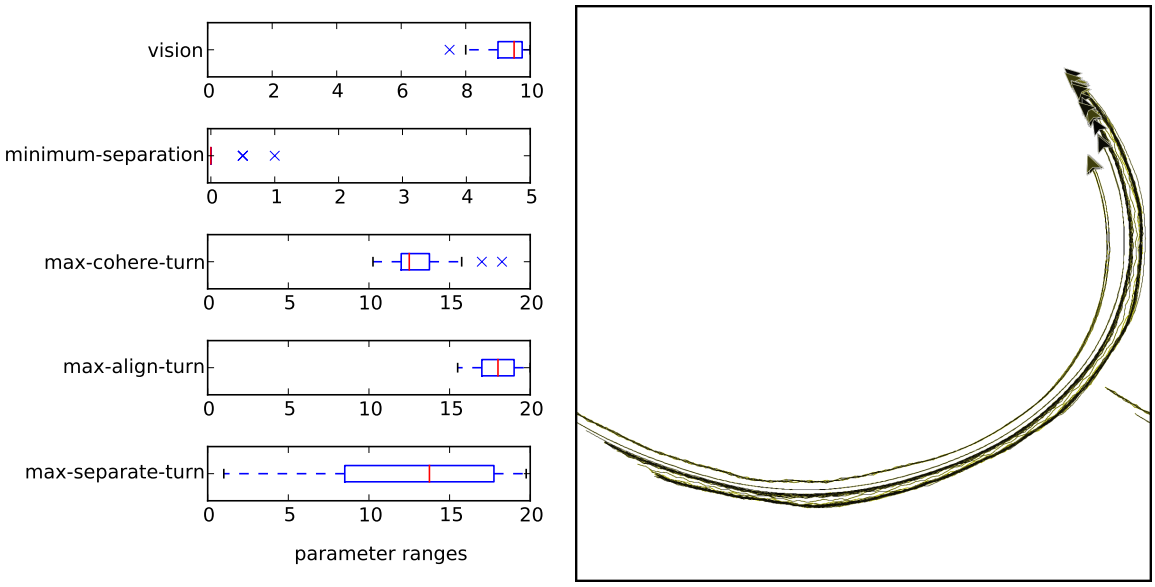


Figure 4.5. *LEFT:* Distribution of model parameter settings (from each of the 30 GA searches) found to cause the most volatility in flock heading. *RIGHT:* Visualization of the flock after 500 model steps (also showing each bird's path over the last 100 steps), using the best parameters found by the GA.

4.4.4. Investigation 4: Vee Formations

The final experiment was to search both the *Flocking* and *Flocking VF* models for a more complex behavior, which we shall refer to as *veeness*. *Veeness* measures the degree to which birds are flying in vee, or more generally, echelon formations. Our specific questions are: 1) Do any parameter settings cause *Flocking* to exhibit *veeness*? 2) How much better can *Flocking VF* do? and 3) What parameters are most important for the best vee/echelon creation?

To calculate *veeness*, we first cluster all the birds in the world into separate flocks, according to proximity (within 5 distance units of another bird in the flock) and directional similitude (less than 20 degrees angular difference in heading). A flock with less than 3 birds is assigned a flock veeness score of 0. Otherwise, it is calculated by choosing the optimal *point*

bird and left/right echelon angles, calculated as described below. (Intuitively, the sum of the two echelon angles is the interior angle of the flock vee.) Echelon angles are constrained to be between 25 and 50 degrees, comprising a mid-range of echelon angles observed in nature [Heppner, Convissar, Moonan Jr, & Anderson, 1985]) for the flock. For any candidate point bird, the left and right echelon angles are calculated separately, by first dividing flockmates into those to the right or left, relative to the point bird. The echelon angles are then chosen such that they minimize the mean-squared-error difference between the echelon angle and the angle between the point bird and all following birds on that side. Flock groupings with echelon angles and flock veeness scores can be seen in Figure 4.8. The flocking score for the flock is the reciprocal of the mean-squared-error value for the best "point" bird, rescaled so that a flock in perfect echelon/vee formation has a score of 1.0. Overall *veeness* is a weighted average (by flock size) of the veeness scores of individual flocks. *Veeness* was measured every 100 model ticks, between 1000 and 2000 ticks. Searches for both *Flocking* and *Flocking VF* used 30 birds and the same veeness metric.

The results show that *Flocking* can create formations that appear only mildly vee-like at best, but *Flocking VF* can (as expected) create much better vees (as shown in Figure 4.8). For *Flocking VF* to produce the best vees (according to our chosen *veeness* metric), the vision-cone angle should be large, perhaps roughly 3 times larger than the obstruction-cone angle, the bird's base-speed and max-turn angle should generally be low, but the speed-change-factor should not be too small. We will not elaborate on specific implications of these findings for the *Flocking VF* model here, but broadly argue that findings such as these can lead modelers to a better understanding of their model by cognitively linking changes in model parameters with the qualitative behavior being investigated.

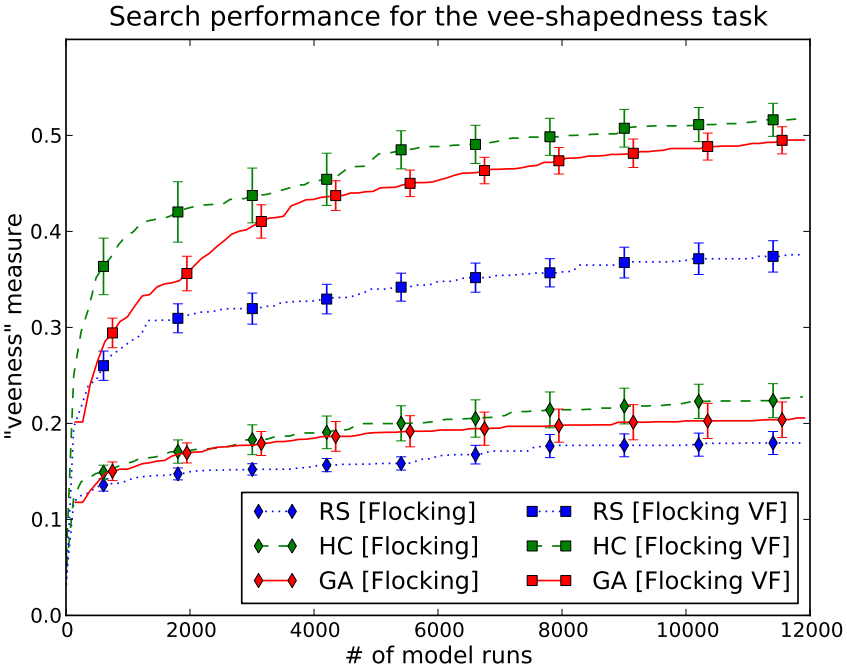


Figure 4.6. Comparison of search performance for the vee-shapedness task on both the Flocking and Flocking Vee Formation models. (Error bars show 95% confidence intervals on the mean.)

Unlike in previous experiments, the HC search method performed slightly better than the GA (see Figure 4.6), but the difference was not statistically significant. For the Flocking model, RS was not far behind GA and HC, indicating that the search space contains a fairly large number of parameter settings that yield a similar level of "veeness" as the best parameter settings that were found (which still are not very good, as we shall see). Given this, it is unsurprising that HC and GA were roughly on par for this task. It is more of a mystery why the GA did not outperform HC in exploring the Vee Flocking model, where good solutions were possible, and both HC and GA significantly outperformed RS. There are a number of possible explanations, but a reasonable hypothesis is that the search space has few local optima where the HC would be trapped, and there is sufficient fitness gradient for the

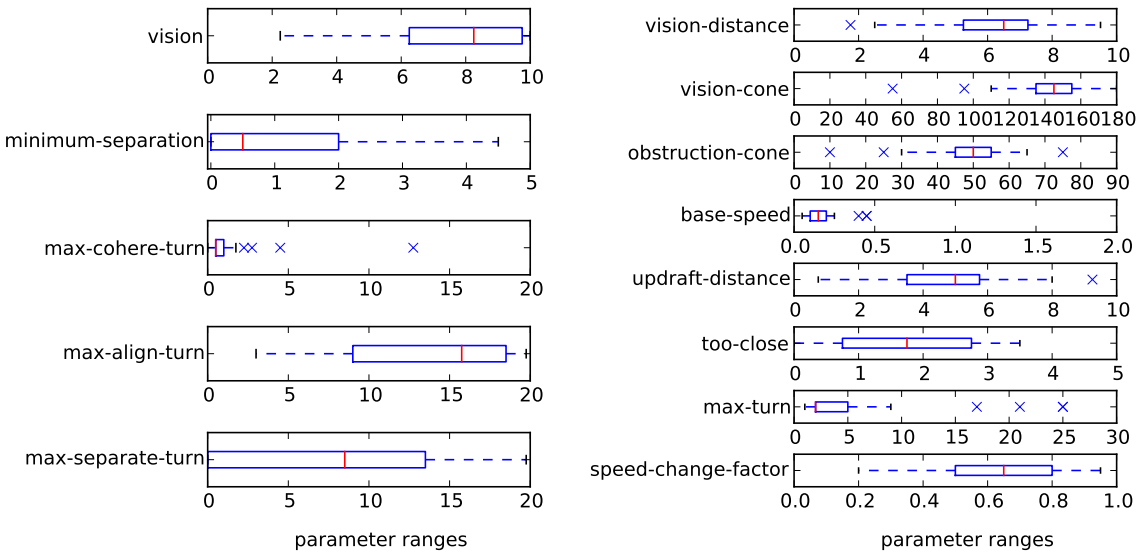


Figure 4.7. Distribution of model parameter settings found to yield the best vees in the Flocking model (*left*), and the Flocking Vee Formation model (*right*), in each of the 30 HC searches.

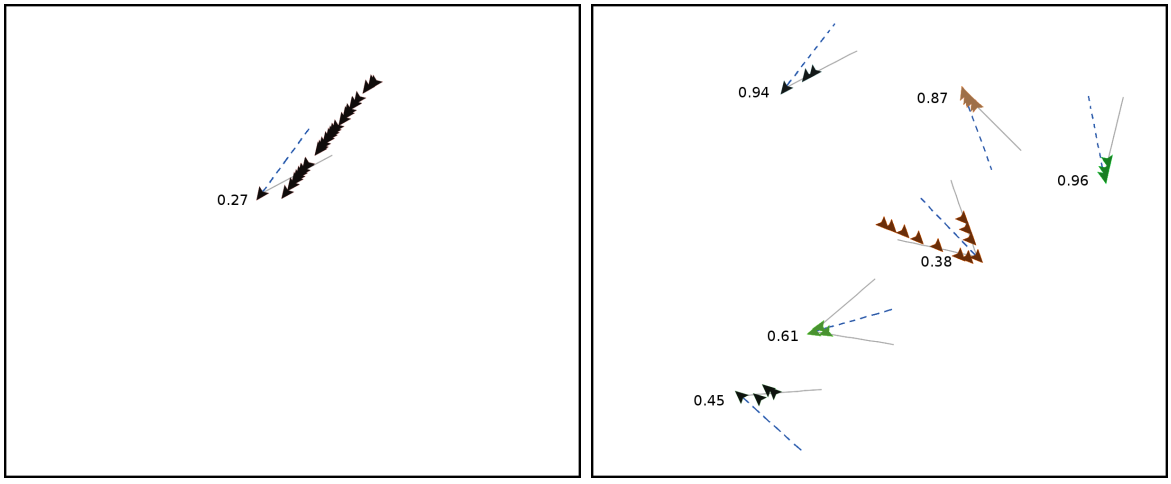


Figure 4.8. Visualization of a run of the Flocking model (*left*), and the Flocking Vee Formation model (*right*), using the best "vee-forming" parameters found by the 30 HC searches. Birds are shaded by flock group, dashed lines show average flock heading relative to the "point" bird, and gray lines show best-fit angles for right and/or left echelons of the vee formation. The numeric "veeness" measure for each individual flock is also shown.

HC to climb to the optimal regions of the space from many initial starting locations. Thus, although the GA is also able to reach the optimal regions of the space, it initially spends more time exploring unproductive areas of the search space in search of better values, rather than climbing directly.

4.5. Conclusion and Future Work

Beyond the specific results concerning the behavior of two particular agent-based models (Flocking and Vee Flocking), there are several more general conclusions that may be drawn from this case study. First, evolutionary algorithms such as the GA and HC are indeed effective means of exploring the parameter space of ABMs. Their performance was vastly superior to RS, except in the cases where the task was too easy (e.g., nonconvergence) or too hard (veeness in *Flocking*) to make substantial progress. The difficulty of the search task relates to how dense or sparse the desired target behavior is in the search space: parameter settings that cause the Flocking model to not converge are plentiful in the parameter space, whereas parameter settings that cause good vee formations are either extremely rare or nonexistent. However, note that the characteristics of the search space could be identical in both of these cases, and whether we classify the task as "too easy" or "too hard" is merely a matter of extrinsically chosen criteria for search success. Second, by running multiple searches on a stochastic model and looking at the distribution of best-found parameter settings, rather than just the single best setting for the parameters, we can uncover trends (or at least postulate relationships) about the interactions between model parameters and behavior. One interpretation is that we are implicitly performing a type of sensitivity analysis on the *search process* for a particular *behavior*, but that the results of that analysis can tell us something about the model. Note that the trends we find are unlikely to be global

(characterizing the whole parameter space), but apply only to a local view that is focused on regions of the parameter space where the target behavior is expressed mostly strongly.

These results also suggest several important areas for additional work. From this single case study, we cannot determine whether genetic algorithms will generally outperform the simpler stochastic hill climbing algorithms for model exploration tasks, or not. This chapter offered a precursory comparison of search algorithm performance on these two flocking models, but Chapter 9 will provide much more extensive performance benchmarking of two types of genetic algorithms (*generational* and *steady-state*) relative to random search, hill climbing, *and* simulated annealing, using a range of ABMs and exploration tasks. We can also conclude from this work that additional consideration should be given to the treatment of model stochasticity and noisy objective functions. While running fewer replicates of model runs takes less time for searching, large quantities of noise can inhibit search progress; this topic will be discussed further in Chapters 8 and 9. In general, this introductory case study shows that the prospects are bright for using meta-heuristic search, such as genetic algorithms, to improve model exploration and analysis. It is our hope that these promising prospects will encourage ABM practitioners to flock toward (and eventually converge on) new methodologies for model parameter exploration that take advantage of these ideas.

CHAPTER 5

Case Study 2: Viral Marketing

"Advertising is the art of convincing people to spend money they don't have for something they don't need."

– Will Rogers

"Marketing is too important to be left to the marketing department."

– David Packard

As an extension of Will Roger's famous quip, we may view "viral marketing" as the art of convincing people *to convince other people* to spend money they don't have for something they don't need. Naturally, this task is not always easy, and figuring out the best way of approaching the problem may be very challenging indeed. Perhaps this is why David Packard believed that the task of marketing should not be entrusted solely to marketing departments, and recent years have shown that computer scientists have something to bring to the table here. Specifically, this chapter will demonstrate that agent-based modeling of interactions on social networks can provide a useful expansion to more traditional techniques of marketing research.

One method of viral marketing involves seeding certain consumers within a population to encourage faster adoption of the product throughout the entire population. However, determining how many and which consumers within a particular social network should be seeded to maximize adoption is challenging. In this chapter we define a strategy space for consumer

seeding by weighting a combination of network characteristics such as average path length, clustering coefficient, and degree. We measure strategy effectiveness by simulating adoption on a Bass-like agent-based model. The Bass diffusion model [F. Bass, 1969; F. M. Bass, 2004] is a well-known model in the field of marketing for characterizing the adoption of a product in the marketplace over time, as the result of interactions between consumers and the twin forces of innovation and imitation. Whereas the classic Bass model assumes perfect population mixing and uses differential equations to predict adoption patterns, the agent-based version applies these principles to local interactions between agents in a social network context [Rand & Rust, 2011]. We examine this model's behavior on five different social network structures: four classic theoretical models (random, lattice, small-world, and preferential attachment) and one empirical (extracted from Twitter friendship data). To discover good seeding strategies, we employ genetic algorithms to search through the parameter-space of agent-based models. This evolutionary search also provides insight into the interaction between strategies and network structure. Our results show that one simple strategy (ranking by node degree) is near-optimal for the four theoretical networks, but that a more nuanced strategy performs significantly better on the empirical Twitter-based network. We also find a correlation between the optimal seeding budget for a network, and the inequality of the degree distribution. A short follow-up study with a second empirical network (an online social network for college alumni) corroborates our findings on the Twitter-based network.

In previous chapters we have noted that exploration is not equivalent to optimization; however, there are many cases where optimization is an important task for model analysis. This chapter focuses on the use of genetic algorithms to perform optimization for optimization's sake, rather than for the purpose of seeking qualitative behavior. However, the particularly interesting findings go beyond merely identifying optimal seeding strategies; it is

the cross-comparison of "optimal" parameters across different model conditions that yields the most fascinating results. Searching a model's parameter space while fixing certain model parameters in turn (e.g., the underlying social network topology and diffusion "virality") can reveal interesting patterns of model behavior. Specifically, in this chapter we can learn about how different viral marketing strategies play out differently in a variety of social networks structures.

5.1. Motivation

Viral marketing, or word-of-mouth marketing, is based on the idea that consumer discussions about a product are more powerful than traditional advertising. One way to encourage positive word-of-mouth is by distributing reduced or free products to target consumers who will then discuss the product with their friends and encourage those friends to buy the product. However, whom to seed with these initial products in order to maximize the amount and rate of product adoption is not obvious. Given an arbitrary social network and a limited seeding budget, choosing the optimal seeding locations has been shown to be an NP-Hard problem [Kempe, Kleinberg, & Tardos, 2003]. Furthermore, it is not clear what the proper seeding budget should be for a particular network. Assuming that the product is beneficial and that seeded consumers are inclined to speak positively about it, seeding more consumers will increase the speed of product adoption. However, giving away more free products increases the overall expense of the promotional campaign. In addition, seeded consumers are removed from the pool of potential customers, which may decrease total revenue for the product. Thus, it is important to choose both the correct target consumers to seed and the correct seeding budget to maximize adoption.

This problem has direct implications for real-world marketing managers. The growth of YouTube, Twitter, Facebook, and other digital social media capabilities, has given marketing managers a new platform by which to advertise and market their products to consumers. The compelling aspect of these platforms is that they encourage consumers to develop online social networks which provide a formalization of the social interactions of individuals. However, despite the power of this new media it has been difficult for marketing managers to use this platform successfully [Baruh, 2009]. In many cases, due to privacy considerations, the full network described by these social media applications is not known, so advertisers are forced to rely on third party information about the consumers they are targeting.

To account for the challenges that marketers face, we propose a version of the general viral marketing problem, which we call the *local viral marketing problem* or *LVMP*. We will first overview related research, then formally define the LVMP, and discuss the agent-based model we use for simulating adoption and the five networks we will test it on. We propose a range of strategies to solve the LVMP, then discuss experimental results from exploring this strategic space using a new evolutionary tool (*BehaviorSearch*), and conclude with recommendations for future work.

5.2. Related Work

Recently there has been work on viral marketing from two different disciplines, computer scientists, and marketing researchers. Originally introduced to computer science by Domingos and Richardson [Domingos & Richardson, 2001], the problem was formalized by Kempe, Kleinberg, and Tardos [Kempe et al., 2003] who described the problem as selecting the correct individuals to seed with a product in an arbitrary network given a fixed marketing budget. They showed that their formalization of this problem is in fact NP-hard,

but presented some heuristic solutions to the problem, with some provable approximation guarantees. However, their best approximation algorithm requires global knowledge of the network; in other words, in order to be implemented the marketing manager would need to know every node in the network and how it is connected to every other node; unfortunately, this is an unrealistic requirement in many real-world cases.

Leskovec, Adamic, and Huberman [Leskovec, Adamic, & Huberman, 2007], on the other hand, take a descriptive approach to viral marketing. Similarly within marketing research, Goldenberg, Libai and Muller [Goldenberg, Libai, & Muller, 2001] use a cellular automata model to describe adoption processes and characterize which individuals have the greatest effect on adoption. Goldenberg and others have also examined the role of hubs (individuals with a high number of friends) in the adoption process [Goldenberg, Han, Lehmann, & Hong, 2009]. Other marketing researchers have explored how innovations diffuse across a variety of different topologies [Shaikh, Rangaswamy, & Balakrishnan, 2006], and how word-of-mouth affects product adoption [Chevalier & Mayzlin, 2006; B. Ryan & Gross, 1943]. In contrast to this previous work, our goal is to make prescriptive suggestions for seeding within viral marketing campaigns, but at a knowledge level that could be available to marketing managers.

To accomplish this task, we use a genetic algorithm (GA) [J. Holland, 1975] to search for optimal (or high-performing) strategies in the space of possible consumer seeding strategies. Our task is equivalent to the problem of optimizing the parameters of a multi-agent simulation, where the parameters control the seeding strategy. In a different context, one of the earliest uses of a GA was to characterize the parameters of a cell simulation [Weinberg, 1970]. Later, Miller proposed the the use of nonlinear optimization techniques for a variety of model exploration and testing tasks, dubbed as "active nonlinear testing" or ANT [Miller,

1998]. Calvez and Hutzler used a genetic algorithm for several parameter search/calibration tasks in an agent-based model of ant food foraging [Calvez & Hutzler, 2005]. Within the marketing domain, Midgley, Marks, and Kunchamwar [Midgley, Marks, & Kunchamwar, 2007] have used a genetic algorithm to examine agent-based models in a consumer retail environment. Building on this research, we have constructed a general tool, which we call *BehaviorSearch* [Stonedahl & Wilensky, 2010a], for using evolutionary computation to explore the parameters of agent-based models created using the NetLogo agent-based modeling toolkit [Wilensky, 1999]. (*BehaviorSearch* will be discussed in further detail in Chapter 10.)

5.3. Local Viral Marketing Problem

The global viral marketing problem[1] (GVMP) consists of selecting a group of individuals who will be seeded with a product in order to encourage their friends to adopt a product at a quicker rate than they normally would have. The problem assumes that there is a graph G, of vertices and edges, where each vertex is a consumer in the network and each edge represents a social connection between two vertices. In addition to the social network, there is also an adoption function, $f_i(t)$, which specifies the likelihood that a vertex, i, will adopt a product at time t, given the adoption state of its immediate neighbors. For the purposes of the results presented herein, the adoption function $f_i(t)$ is assumed to be the same for all individuals, so we will use the notation $f(t)$.

In order to simultaneously consider both the amount and rate of adoption, we will use the notion of the net present value (NPV) of an adoption network [Goldenberg, Libai, Moldovan, & Muller, 2007]. Intuitively, the NPV measure accounts for the fact that it is worth more to a company if people buy its product now, rather than several months from now, especially

[1]Also referred to as the Influence Maximization problem in some contexts.

since new competing products may enter the marketplace. The NPV, given an adoption function ($f(t)$), social network (G), and seeded vertices (S), is the sum of vertices that adopt the product multiplied by the profit from the product and a discount factor for time of adoption, specifically:

$$NPV(G, S, f(t)) = \sum_{t=0}^{\infty} a(t) p \lambda^t$$

where $a(t)$ is the number of adopters at time t, p is the profit for adoption of a product, and λ is the discount factor. In our experimental results, we chose a 10% discount rate ($\lambda = 0.9$), which has previously been used in related marketing literature [Goldenberg et al., 2007; Libai, Muller, & Peres, 2009]. This discount rate represents the cumulative effect of several factors, including the opportunity cost of not having the money earlier and the potential necessity to lower prices over time to stay competitive. [2] The fully specified GVMP is to identify a set of vertices S that will maximize the network's NPV, given that $|S| \times c \leq b$, where c is the cost of seeding one vertex, and b is a specified budgetary constraint. In the terms of the QBME framework from Chapter 3, the model behavior we are interested in is the aggregation of the number of agents whose individual-level state changes (from "not adopted" to "adopted"), condensed over time. However, the NPV measure has the interesting property that it doesn't average equally across time steps - instead it is a weighted average over time, where events that happen later are given less weight.

The local viral marketing problem (LVMP) is similar to the GVMP, except that we remove knowledge of the structure of the global network (G), instead offering only characteristics of each vertex which provide summary statistics about the vertex and its role in the network. There are many different network measures that could be chosen [Wasserman &

[2]Preliminary comparison experiments suggest that using other reasonable discount values is unlikely to change our qualitative results.

Faust, 1994]; in Section 5.4.3 we will describe the specific measures we used, but one example measure is the vertex degree (i.e., the number of neighbors). Specifically, the problem is to find a weighting function, $w(i)$, that determines where to place vertex i in a priority queue for seeding. Once the queue has been created, vertices to be seeded are chosen in rank order, until the budget (b) is exhausted. Also, in contrast to the GVMP, in our formulation of the LVMP we allow b to be varied as part of the strategy, which includes finding an optimal budget amount as part of the problem definition. Thus, we define a seeding strategy, S, to be a weighting function $w(i)$ together with a specified budget, b, as this is sufficient information to seed an arbitrary network.

Our examination of the LVMP is arguably more relevant to the real-world than the GVMP for a number of reasons. As discussed in Section 5.1, often the best budgetary value to use for viral marketing seeding is unknown. Moreover, in many real-world cases the global social network is also unknown. In face-to-face interactions, no one knows the full network of any reasonably sized market, and even in the case of social networking web sites, privacy constraints may prevent access to the whole network (e.g., Facebook), or data collection limitations may be prohibitive (e.g., Twitter). Even in cases where data is available, running simulations on the entire network to determine the optimal seeding strategies would be computationally difficult, if not impossible. Solving the LVMP for realistic networks of moderate size could provide marketing managers with a way to specify solutions that are not reliant on global network knowledge. Moreover, since the LVMP strategies are specified in a generalizable way that is not dependent on a particular network structure, they may facilitate learning of solutions that perform well across a variety of network architectures. Finally, solutions to the LVMP could be used to drive new business models. If the role of an individual in diffusion is known, then social media platforms, such as Facebook, or intermediaries who

work with these platforms, such as a third-party advertising firm, could charge different premiums to brands for advertising to different types of consumers, based on the consumers network characteristics. For instance, they might charge more for an advertising campaign targeting well-connected users than for a campaign using random sampling. Solutions to the LVMP would provide a way to quantify the differential utility, and appropriately price these campaigns.

In this chapter, we specifically address these questions: How do different social networks affect the optimal seeding budget and strategy? Does providing a complex strategy space yield better solutions than simple strategies? How robust are LVMP strategies to different adoption "virality" levels?

5.4. The Model

In order to investigate the LVMP, we must specify a model for the diffusion of products throughout the network. Specifically we must describe an adoption function, $f(t)$, the network structure, G, and the strategy space, S.

5.4.1. Adoption Function

There are at least two classes of product adoption function that have been examined, Bass-like models [Rand & Rust, 2011; Goldenberg et al., 2001] (sometimes called "cascade" models), and "threshold" models [Watts, 2002; Watts & Dodds, 2007]. In the Bass-like model (so-called because of its resemblance to the aggregate-level Bass model [F. Bass, 1969]), the adoption decision consists of two factors, whether to adopt due to individual innovation, and whether to adopt due to peer imitation. In a "threshold" model, each individual adopts only if the fraction of their neighbors that has adopted is above a certain threshold. We will use

a Bass-like adoption function that is the most immediate translation of the aggregate Bass model to an individual level and is an example of an independent interaction model that has been previously examined in similar forms [Goldenberg et al., 2009; Watts & Dodds, 2007; Shaikh et al., 2006; Toubia, Goldenberg, & Garcia, 2008]. In our model, the heuristic for adoption of individual i can be written as, $f(t) = p + q(\frac{n_a(t)}{n})$ where p is the effect of external influences on adoption, q is the effect of social influences on adoption, n is the number of neighbors of i, and $n_a(t)$ is the number of neighboring vertices who have already adopted the product at time t. Although this adoption function clearly does not capture all aspects of real-world influence between consumers, it has been validated against empirical data with good results [Rand & Rust, 2011].

In the present work, we examine two different diffusion scenarios: a 'medium virality' scenario ($p = 0.01535$ and $q = 0.455$) and a 'high virality' scenario ($p = 0.0007$ and $q = 0.53$), which are at the middle and extremes (respectively) of empirically observed values [Chandrasekaran & Tellis, 2007]. We do not examine a 'low virality' scenario (high p and low q), since the dominance of individual adoption over peer-based word-of-mouth minimizes the network-effects that interest us, and viral marketing does not significantly affect adoption.

Different values of p and q may be seen to represent different types of products. A product which has a high p relative to other products is one that consumers will naturally adopt on their own; this could represent a product which is just clearly useful, such as a refrigerator. A product which has a high q relative to other products is one which consumers are more likely to adopt if many of their friends have adopted; this could represent a product with considerable network efforts, such as a fax machine, or a product which encourages social discussions, such as the Flixster Facebook app for sharing movie recommendations.

5.4.2. The Networks

In the experimental results below, we investigate four abstract networks created using network generation routines from the social network literature, along with one empirically derived network. In all five cases, the number of nodes [3] in the network is exactly 1000. For the generated networks, we also chose parameters that would yield a similar[4] edge density to that of the empirically derived network. Specifically, the networks are:

(1) **random** - an Erdös-Renyi random graph [Erdős & Rényi, 1960], with a uniform probability ρ of an edge being present between any two vertices ($\rho = 0.26\overline{712}$ in the results below).

(2) **lattice** - a regular network, where each node in the network is located on a circle and connected to a particular number of neighbors (26 in the results below) on either side of them.

(3) **small-world (sw)** - this network is generated by starting with a lattice network, and randomly rewiring some of the edges as described in [Watts & Strogatz, 1998] (in the results below, we used a degree of 26 and a rewiring probability of 0.01).

(4) **preferential attachment (pa)** - this network is generated with the preferential attachment mechanism described in [Barabasi & Albert, 1999]. Nodes are incrementally added to the network and connect in a way that is preferentially biased toward individuals who already have many connections (in the results below, 14 connections created per added node).

---

[3]The terms *graph*, *vertex*, and *edge* come from graph theory, whereas *network*, *node*, and *link* are often used in network science – we will use these terms interchangeably.

[4]Matching the exact number of edges is not possible with these network generation algorithms, but reasonably similar edge densities were obtained.

(5) **twitter** - this network was extracted from data available via the public Twitter API. It represents a small connected subgraph of the complete Twitter social network. Starting with a random Twitter UID between 1 and 10 million, we used breadth-first search to add the 999 nodes closest to our starting node, and all friendship links ($13,343$ in this case) between these nodes. (Note: we define A and B as friends when A "follows" B *and* B "follows" A)[5].

Visualizations of the five networks are shown in Figure 5.1). The *lattice* and *random* networks are not realistic social networks, but they are used for comparison purposes, as well-studied examples of extreme order and disorder (respectively). The small-world (*sw*) and preferential attachment (*pa*) networks have been shown to model certain types of social and constructed networks fairly well [Barabasi & Albert, 1999; Watts & Strogatz, 1998]. The *sw* network has a high level of clustering, while maintaining a short average path length. The *pa* network exhibits a power law (or scale-free) relationship between the degree of nodes and their frequency of occurrence. The *twitter* network provides an example from a real digital social network. It displays a more skewed degree distribution than even the *pa* network, indicating that a very small number of individuals have a disproportionately large number of social connections.[6]

5.4.3. Strategies

In order to evolve solutions for the $LVMP$, we need to define the search space for optimal strategies. In Section 5.3, we define a strategy to consist of two elements: the budget b, which

[5]In this work, for simplicity and consistency, we used only undirected networks. However, this approach ignores the potentially important effects of assymetric following relationships on Twitter, and future work should include additional investigation using directed networks

[6]This may partially be an artifact of our subgraph extraction method, but the degree distribution of the complete Twitter network is likely to be similarly skewed.

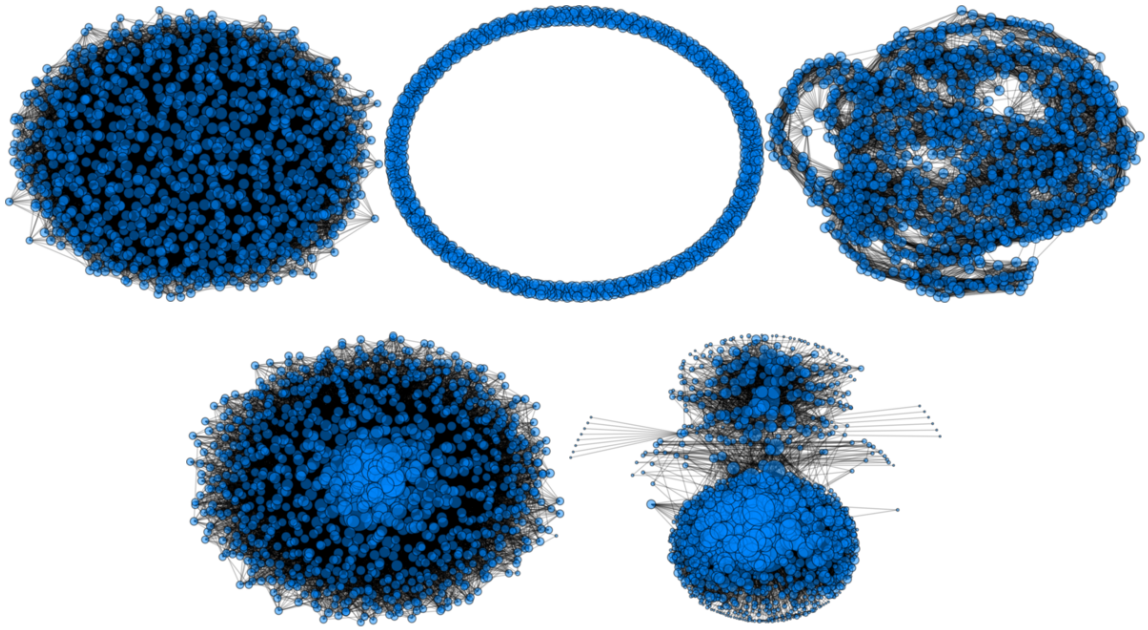


Figure 5.1. Visualization of the *random*, *lattice*, *small world (sw)*, *preferential attachment (pa)*, and *twitter* networks (listed in left to right, top to bottom order). The size of each node illustrates its degree (number of neighboring nodes) in the network.

we will operationalize as the fraction of the total network to be seeded, f_s, and a priority weighting function $w(i)$. For the experiments presented here, we will assume no additional cost c for seeding a node beyond the loss in potential profit p that would otherwise have been gained from a node if it had adopted, thus the budget cost b is reflected in the ineligibility to adopt the product of the initial $f_s \times n$ seeded nodes (where n is the size of the network). This is a generally optimistic view of seeding costs, but may be realistic for digital media products, where after the sunk development costs, the marginal production cost is near 0. A useful weighting function for determining seeding priority requires information about individuals. In this work, we will assume knowledge is available about several characteristics of nodes in the network, illustrated by the following five simple weighting functions:

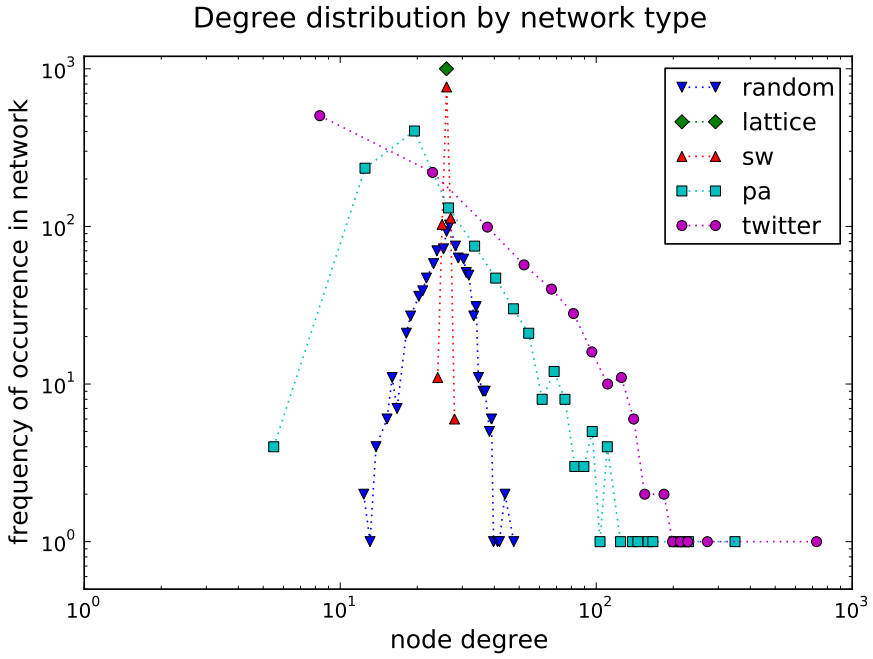


Figure 5.2. Degree distributions for each network, displayed on a log-log plot. As the precise shape is dependent on binning choices, this histogram is meant only to give a general sense of the degree distributions. The dotted lines serve only to guide the reader between data points.

(1) *degree* - the number of neighbors of the target node normalized by the maximum possible value, i.e., $w_d(i) = \frac{degree(i)}{max(degree)}$. Higher degree nodes influence more neighbors, directly encouraging more adoption.

(2) *twostep* - a measure corresponding to the number of nodes that are reachable within two steps of the given node (by following edges in the network), $w_t(i) = \frac{twostep(i)}{max(twostep)}$. An extension to the degree measure.

(3) *average path length (apl)* - the average number of steps from this node to any other node subtracted from the maximum average path length of any node in the network

and normalized: $w_a(i) = \frac{max(apl)-apl(i)}{max(apl)}$. Nodes with lower average path length are better connected to the entire graph, potentially encouraging adoption.

(4) *clustering coefficient (cc)* - 1.0 minus the fraction of neighbors of the node whose neighbors are also neighbors of the target node, normalized by the highest clustering-coefficient in the network, i.e., $w_c(i) = 1.0 - \frac{cc(i)}{max(cc)}$. The lower the clustering coefficient of a node, the less overlap there is among its neighbors, encouraging wider adoption more quickly.

(5) *random* - the priority of an individual is determined randomly, i.e., $w_r(i) = U[0, 1]$.

(Note that each weighting function is normalized so that values fall within the range of $[0, 1]$, and higher values of $w(i)$ will correspond to a better ranking in the priority queue, and that ties will be broken randomly.) In past work on the GVMP, the degree and apl have been shown to be important factors, while random seeding performs poorly [Kempe et al., 2003].

We hypothesize that better solutions (using the same available information) than these 5 simple strategies may be possible if the strategies are employed together. Thus we consider weighting functions that use a linear combination of the strategies above:

$$w_{comb}(i) = \alpha_d w_d(i) + \alpha_t w_t(i) + \alpha_a w_a(i) + \alpha_c w_c(i) + \alpha_r w_r(i) \tag{5.1}$$

where the α's express the normalized weights assigned to each of these various characteristics of the node. Finally, a linear combination might still not be expressive enough; what if it were better to alternate seeding between two different strategies? For instance, first seed the highest degree node, then the node with the lowest normalized path length, and back and forth until the budget is exhausted. Therefore we expand our space to include "mixed strategies", consisting of two sub-strategies, along with an additional parameter for how

often each substrategy should be used. This gives us our final $w(i)$ function, which is:

$$w(i) = \begin{cases} w_{comb,1}(i) & \text{if } x < p \\ w_{comb,2}(i) & \text{otherwise} \end{cases} \tag{5.2}$$

where $w_{comb,1}$ and $w_{comb,2}$ are both of the form described in Equation 5.1 with their own α's, x is a random variable drawn[7] from $U[0,1]$, and p is the parameter which specifies the probability with which $w_{combined,1}$ is to be used. Without loss of generality, we restrict $p > 0.5$, meaning that $w_{comb,1}$ will always be the primary sub-strategy and $w_{comb,2}$ is the secondary sub-strategy (chosen less often for seeding). Given this space, we can now describe an individual in the population of our genetic algorithm. Each individual will specify weights for all the α values described above (ten different values, five for each of the two strategies), a p which is the probability with which the first strategy is used, and f_s which is the fraction of the population to seed. This results in 12 real-valued genes for each individual, which is not especially many, yet the search space is too large for a brute-force approach. Also, given the complexity and stochasticity of the fitness function, we speculate that the space will be highly nonlinear, and there will be noise in the fitness determination (discussed below). These factors motivate our choice of genetic algorithms for exploring this problem.

5.5. Implementation

In order to explore the LVMP, we constructed an agent-based model of it using NetLogo [Wilensky, 1999]. A screenshot of this agent-based model is shown in Figure 5.3. In the model, we first create a number of agents (1000 for the experiments presented herein), and then we connect them according to one of the social network topologies described in Section

[7] x is only drawn once per seeding choice

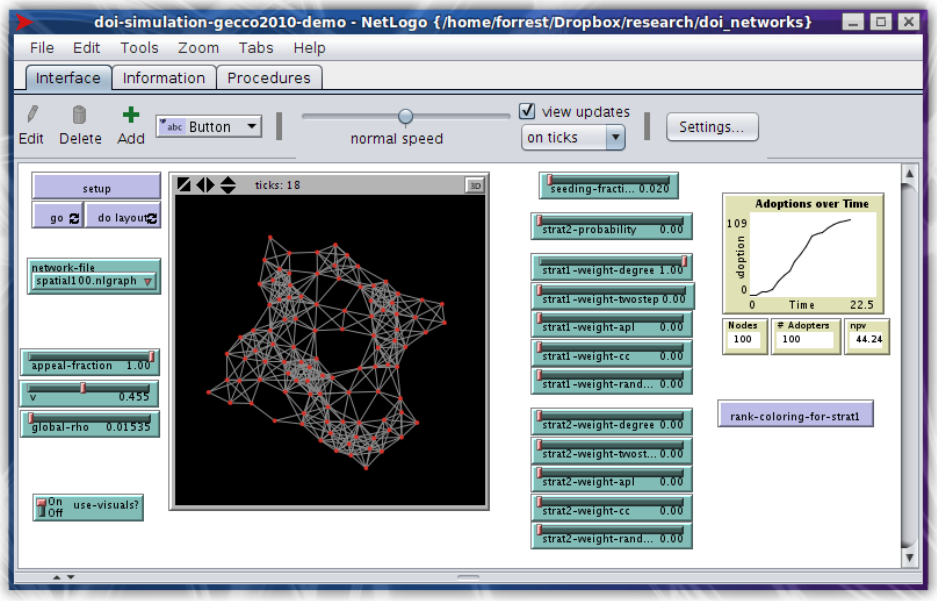


Figure 5.3. The agent-based model of product adoption in a social network setting, implemented in NetLogo. The parameters on the left side of the model interface were held constant during a single GA search, whereas the parameters on the right side of the interface (which control the initial seeding strategy), were evolved by the GA.

5.4.2. Then we take the strategy currently being investigated, and we sort the list of all agents using Equation 5.2. After this we select the fraction of agents at the top of this priority queue using the f_s specified by the strategy, and we seed each of these agents with the product (setting their adoption state to true). Then at each time step of the model, every agent who has not adopted the product runs the decision rule described in Section 5.4.1 to decide if they will adopt the product. Once all the agents have decided whether to adopt the product in a particular time step, we record the total number of consumers who have adopted and we begin the next time step. In our experiments, we make the simplifying assumption that the product has some appeal to every agent in the population, thus the simulation ends once all agents have adopted, and we calculate the NPV of the current

run. Since the adoption heuristic is stochastic and the seeding strategy may be stochastic, we run the simulation multiple times to more accurately calculate the expected NPV for a given strategy. Specifically, an individual's fitness is the average NPV from 10 simulations (with different random seeds). While this Monte Carlo averaging cannot eliminate noise, in practice we found that using 10 replicates sufficiently reduced the noise so the GA could progress toward good solutions. Moreover, GAs are often successful despite the presence of noise or uncertainty [Jin & Branke, 2005].

To automate the process of exploration, we have created a tool called *BehaviorSearch* [Stonedahl & Wilensky, 2010a] that interfaces with NetLogo, and which can run a genetic algorithm over the parameters of any NetLogo simulation. In this case, the parameters of our model correspond to the seeding strategy to be evaluated. The genetic algorithm used is reasonably simple: we generate an initial population consisting of 50 random individual strategies, each containing 12 different genes as described in Section 5.4.3. The numeric values that make up a strategy are discretized at a resolution of 0.01, and encoded as a binary string, using a Gray code[8].

The fitness of each individual is evaluated by decoding the binary string into the 12 strategy parameters, initializing the agent-based model with these parameters, and observing the mean NPV from 10 independent replications of the simulation. Using these fitness values, BehaviorSearch performs a standard generational GA [J. Holland, 1975] evolution step (70% one-point crossover, 1% mutation rate, tournament selection with tournament size 3) on the population. This process is repeated for 200 generations. For both the 'medium' and 'high' virality scenarios, we used BehaviorSearch to conduct multiple instances (30) of

[8]We chose this representation since prior research suggests that Gray binary encodings are superior to traditional high-order bit encodings [Caruana & Schaffer, 1988; Whitley, 1999], and our own preliminary comparison experiments using real-coded genes did not appear advantageous to the Gray encoding.

these searches on each of the five different networks (Section 5.4.2); this resulted in a total of 2 *scenarios* $\times$ 5 *networks* $\times$ 30 *searches* $\times$ 50 *individuals* $\times$ 200 *generations* = 3 *million fitness evaluations*. As each fitness evaluation requires averaging 10 runs, the grand total is 30 million simulation runs, which took approximately 11,000 hours (or 462 days[9]) of compute time.

5.6. Results and Discussion

The first result we will examine is the GA's performance across the different networks types. Figure 5.4 shows the best-of-run performance for the GA on each network topology, for the 'medium virality' scenario (performance trends for the 'high virality' scenario were very similar). The GA finds fairly good solutions for each topology early on and then the rate of improvement slows after that. The effect of noisy fitness evaluation is observable, in that the actual NPV values (dotted lines in the figure, approximated by the average NPV from 1000 simulation runs with the GA's best individual) are considerably lower than the best-of-run fitness values the GA reports (solid lines). This is because the GA only averages 10 simulation runs to determine fitness, and then it chooses the best from the population, so the noise causes an overly-optimistic estimate of the best individual's fitness. However, individuals with the highest noisy fitness are likely to also have highest actual NPV, and the correlation between the increase in fitness and the increase in the actual NPV confirms that the GA does make real progress despite the noisy environment. Figure 5.4 also demonstrates that there are different maximum NPV values achievable for each social network. In fact, there is substantial variability in the capacity of these different networks to transmit/diffuse information which directly affects NPV. In general higher NPV values were possible on the

[9]Less than a month in real-time because these searches were distributed across a computing cluster.

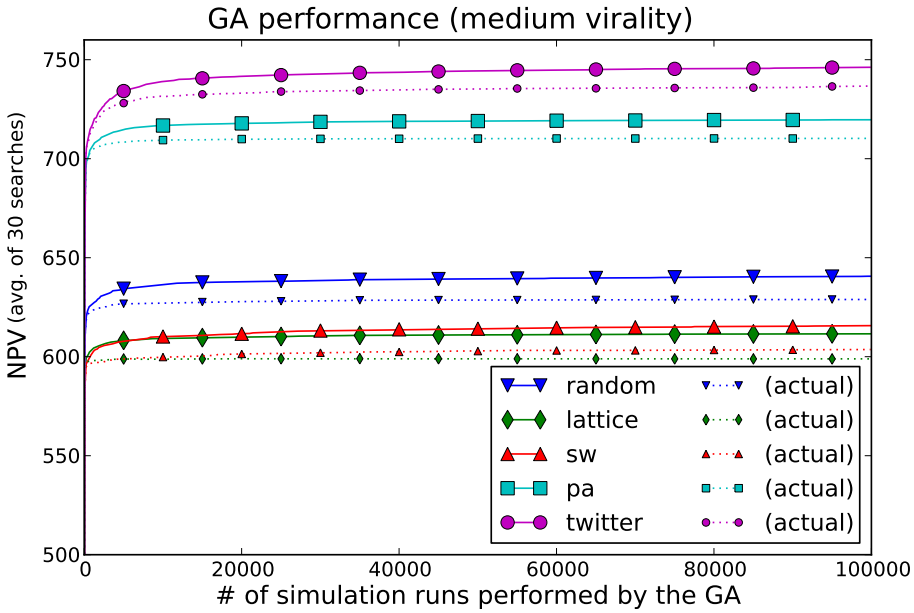


Figure 5.4. GA progress (averaged across 30 searches) by network topology, for the 'medium virality' scenario. GA's reported best-of-run fitness (solid lines) are compared with the actual NPV values (dotted lines), estimated by 1000 simulation runs, showing the effect of noise. (Error bars too small to show.)

networks with degree distributions that were more skewed, or inequitably distributed (in particular, the *pa* and *twitter* networks). The NPV values have a theoretical maximum of 1000 (unattainable), which would correspond to every person spontaneously deciding to adopt the product immediately, without any seeded individuals.

Before examining the evolved strategies, we will discuss results for the seeding budgets (seeding fraction, f_s) discovered by the GA. In all of the 30 search replications, the chosen f_s was always centered tightly around a specific value, which indicates a high degree of confidence that the seeding fraction values that were found are indeed optimal. However, the specific value of f_s varied substantially between network types, and also slightly based on the virality scenario (see Figure 5.5). In general, f_s was lower for those networks with degree

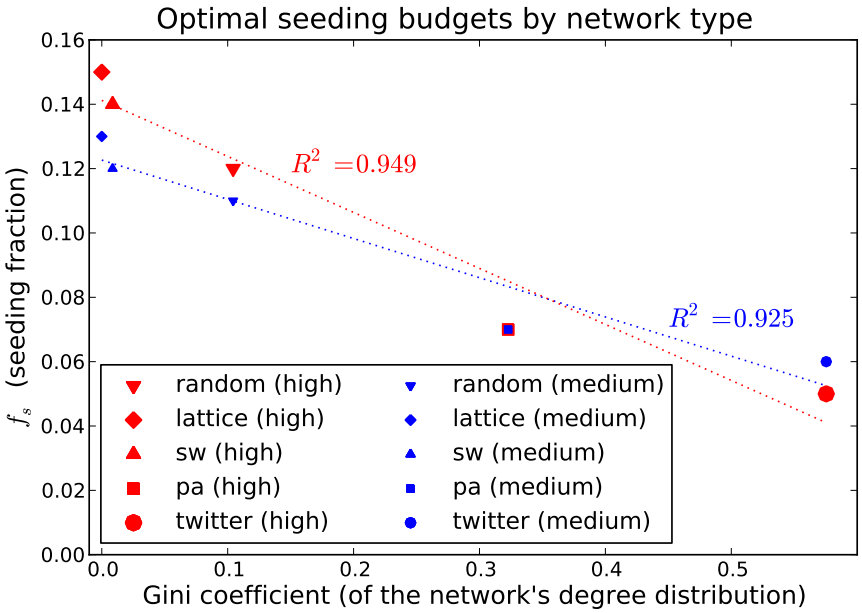


Figure 5.5. The best seeding budgets found by the GA for each network type. These are plotted against (on the x-axis) the Gini coefficient of the degree distributions. The regression lines are not intended to propose a linear relationship, but merely to illustrate the correlation.

distributions that were skewed such that a small number of nodes had a disproportionately large number of connections. Figure 5.5 displays this relationship quantitatively by plotting the optimal seeding budgets (as discovered by the GA) against the Gini coefficient [Gini, 1912] of the network's degree distributions, which is a standard measure of distributional inequality ranging from 0.0 (flat equal distribution) to 1.0 (all connections concentrated in a single individual). This relationship also mirrors how the maximum achievable NPV varies by network type: essentially networks with uneven degree distributions have lower optimal seeding budgets, and a higher payoff in terms of adoption (NPV). This result is sensible, given that degree (w_d) turns out to be very important component of seeding strategies for all of the networks, as we will discuss below.

The next question we investigated was what the best strategies discovered by the GA looked like. For each of the 5 network types, in each of the 2 virality scenarios, BehaviorSearch provides us with the best strategies found in each of 30 GA searches. We present here the results for the *twitter* network (which proved to be the most interesting case) on the 'medium virality' scenario (see Figure 5.6). As shown, there is a fair amount of variation among the GA's best strategies. This is likely due to large plateau areas in the landscape resulting in neutral evolution among a variety of different strategies, though it could also indicate non-convexities in the space that make it difficult to search. Relating this back to the behavior of the agent-based model, this result suggests that the rate of adoption throughout the social network is not very sensitive to changes in some of the parameters controlling seeding strategy. When using genetic algorithms to search the parameter-space of agent-based models, the diversity of the parameters returned by the search can be informative about the model, as discussed in the QBME framework (see Section 3.4.5). It is worth emphasizing that the weight given to the "random" strategy (α_r) was very low in all the best strategies found by the GA (except for in the *lattice* network, where no LVMP strategy outperforms random because all nodes have identical characteristics), which shows that choosing an informed strategy for seeding is important.

Our next inquiry was whether the GA's best strategies gave better performance than using very simple strategies with the same available information. For each set of 30 strategies generated by the evolutionary search, we determined the "best strategy" by testing them with an additional independent 1000 simulation runs, and choosing the one with the highest average NPV. As a baseline for comparison, for each network, we also determined an NPV value by seeding using each of the basic component weighting functions individually: degree

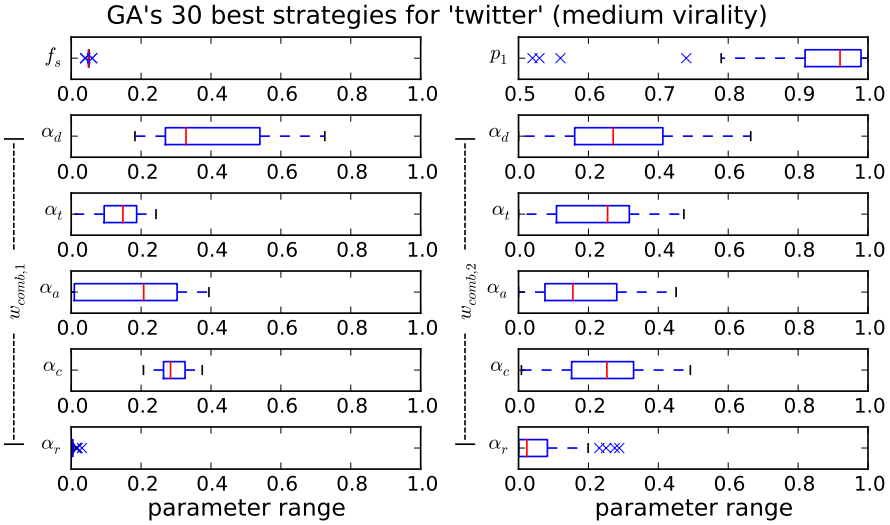


Figure 5.6. Box and whisker plots showing the variation among parameters for the best strategies that the GA found for the *twitter* network ('medium virality' scenario). These strategies' NPV performance varied slightly but was consistently high (from 733 to 741). (Boxes show middle quartiles with median marked red, and outliers as ×s.)

(w_d), two-step neighbors (w_t), average-path-length (w_a), clustering-coefficient (w_c), and random (w_r). On each of the five network types, w_d proved to be the best basic strategy of the five basic strategies. On four of the five networks, the best strategies found by the GA were either only very marginally better, or not significantly different than w_d, with the notable exception being the *twitter* network, where the GA found a strategy that outperformed w_d by more than 19 NPV units, or 2.5% ($p < 0.01$ significance), in both the 'medium' and 'high' virality scenarios. A performance comparison is shown in Figure 5.7, and the GA's best strategies for the *twitter* network are displayed in Figure 5.8. Further testing showed that in the best strategy for the 'high virality' scenario, the 1% use of a secondary strategy had no impact, and in both 'medium' and 'high' scenarios, the small amount of α_a (apl) weighting included in these strategies was not significant in affecting performance. Thus, the key strategic ingredient turned out to be the combination of high degree (α_d) with low

clustering-coefficient (α_c). This is interesting, since using w_c (cc) alone as a weighting strategy performs worse than random seeding on the *twitter* network. The poor performance of w_c comes from a sizable number of degree one nodes (only a single friend) in the *twitter* network, which (trivially) have clustering coefficients of 0, but make poor choices for seeding. These findings beg the question: what is special/different about the *twitter* network, that was not captured in any of the 4 abstract generated social networks, which makes clustering coefficient information important for seeding? Our hypothesis is that many of the highest degree nodes (hubs) in this *twitter* network are closely linked with one another, but that there are some important individuals in the network that are further away from the central hubs, and serve as "brokers" to individuals or groups that are not directly connected to the hub. The visualization of seed choices within the *twitter* network (Figure 5.9) supports this explanation. Logically, it makes sense to seed individuals that are both reasonably high degree, and also play the role of brokers in the network – and yet, the $w_d + w_c$ combination does not outperform pure w_d on the other four networks. This might indicate that the four artificial networks fail to capture an important component of real social networks.

One of our research questions was whether mixed strategies (i.e. alternating between two sub-strategies) offered any advantages over pure (single) linear-combination strategies. In our current results we do not see any benefit, as the GA was only able to find a strategy that outperformed the simple w_d degree strategy in the *twitter* network, and that turned out to be a pure strategy as well, only requiring a combination of w_d and w_c to succeed. However, this does not rule out the possibility that a mixed strategy could be useful with a different network from the 5 investigated here, or with a different set of available network characteristics.

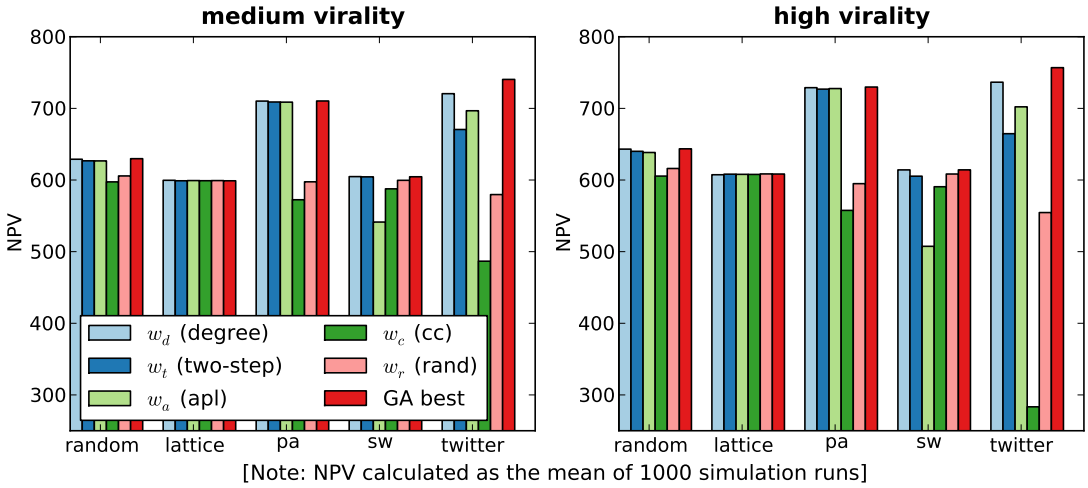


Figure 5.7. Best strategies found by the GA compared against the 5 basic component strategies.

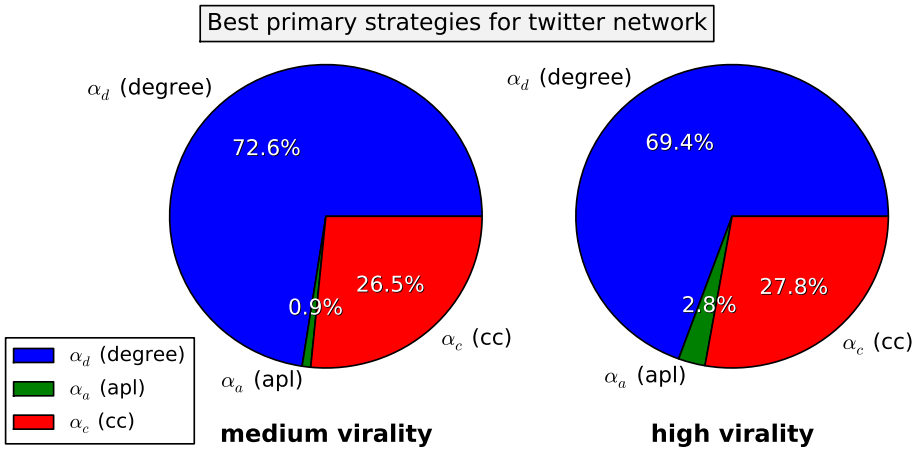


Figure 5.8. Components of the best primary sub-strategies the GA found for the *twitter* network. Secondary sub-strategies were basically unused: $p_1 = 1.00$ ('medium') and $p_1 = 0.99\%$ ('high').

All evidence so far suggests that LVMP strategies are robust across different "virality" levels. In particular, the simple w_d strategy performed fairly well across the board, and the improved $w_d + w_c$ combination strategy for the *twitter* network was very consistent between the 'medium' and 'high' virality scenarios. This is a hopeful sign for marketing managers, in

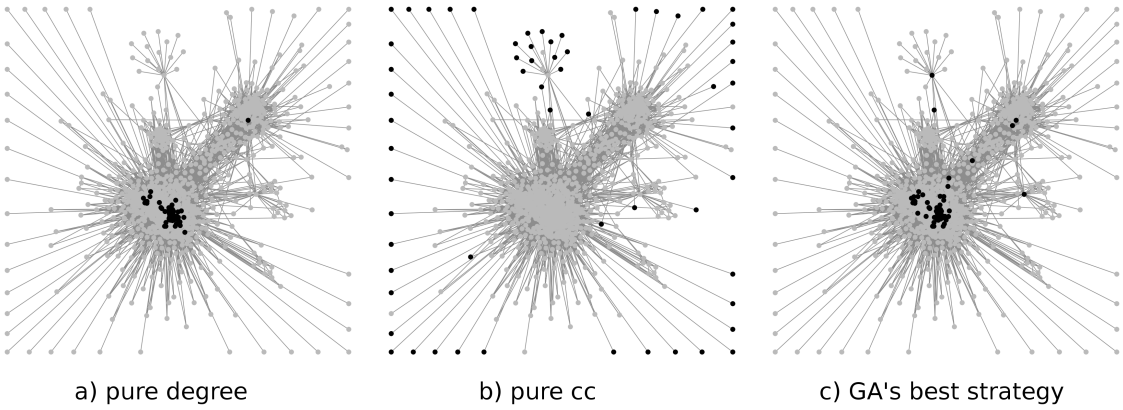


Figure 5.9. Visualization of three seeding strategies on the *twitter* network.

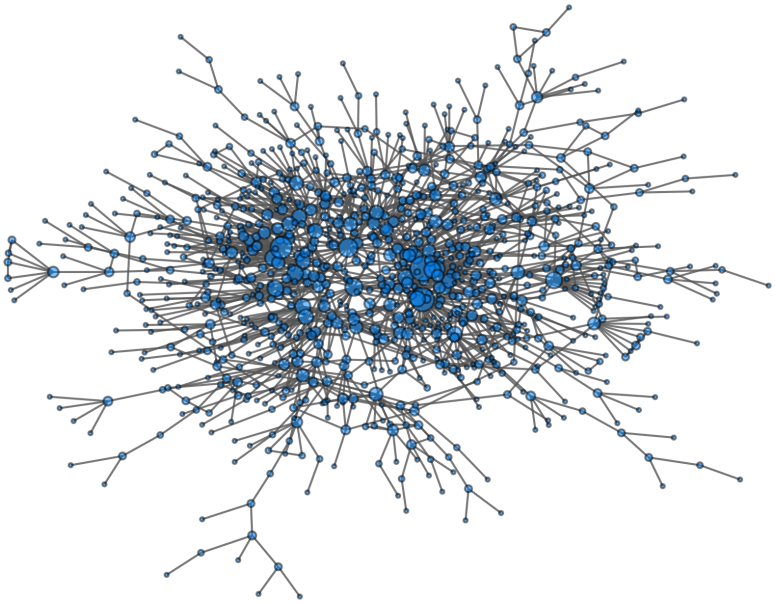


Figure 5.10. Visualization of the *alumni* network used in the follow-up experiment as a second empirically-based social network.

that results may be generalizable across different types of products, and (to a lesser extent) across different social network structures.

5.7. Follow-Up Experiment on the Alumni Dataset

To either support or dis-confirm our hypothesis regarding the difference between abstract and empirical networks, we obtained a second empirical social network and performed a follow-up experiment. In this case, our data source was a small online social networking site for college alumni, with the largest connected component containing 938 users and 1399 links between users. Each link represents a symmetric social tie, based on a threshold number of combined communication interactions (email, chat, leaving a messaging, etc) going in both directions between the users. A visualization of this network is provided in Figure 5.10. Note that this network is significantly different from the *twitter* network, both in terms of what it represents (discretized social interactions on a social networking site versus "mutual followers" on a social *media* network) as well as the network properties. Primarily, the *alumni* network is much less dense (average degree of 3.0, as opposed to average degree of 26.7 for *twitter*). The *alumni* degree distribution is highly skewed, although not quite as unequally distributed as the *twitter* network (Gini coefficient of 0.44 versus 0.58 for *twitter*).

However, despite the differences, the best seeding strategies for the *alumni* network (discovered by performing genetic algorithm searches similar to those described above) were strikingly similar. Again the best strategy was a combination of weighting by degree and clustering coefficient. Specifically, both the medium and high virality scenarios yielded around two-thirds weighting by degree and one-third by clustering coefficient (the precise breakdown for both cases is shown in Figure 5.11). Although these numbers don't perfectly match the *twitter* strategy, this follow-up investigation strongly corroborates two qualitative results from the original study: 1) that using a combination of clustering coefficient and degree is beneficial for selecting seeds for the LVMP, and 2) that there is something different about

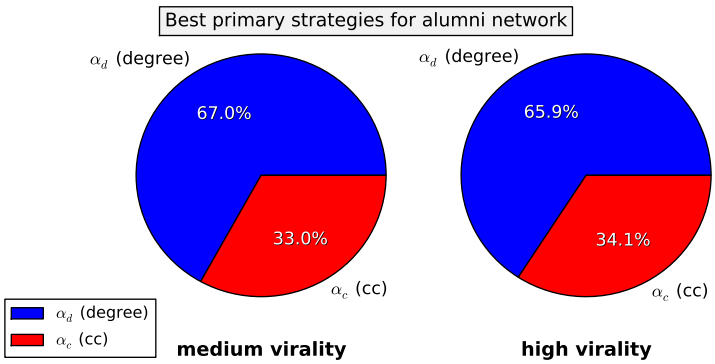


Figure 5.11. Components of the best primary sub-strategies the GA found for the *alumni* network.

empirical networks (possibly due to community structure) which is not being captured by the abstract network generation models.

5.8. Side Note on Search Performance

Our primary focus in these experiments was on obtaining good solutions (strategies) for the LVMP domain, a task for which the genetic algorithm proved very effective. However, there raises the further question: *very effective relative to what*? First, we note that an exhaustive factorial (grid-sweep) experiment of the parameters at this fine-grained resolution is simply not feasible. In fact, even at very low resolution (say, 3 settings per parameter), simulating that number of combinations would take about 50 times longer than a single GA search at the fine resolution. However, simpler meta-heuristic search algorithms than GAs exist, and it is valid to question whether GAs provide better performance in this task. Arguably the most basic search algorithm is random search, which samples points uniformly at random from the parameter space. Random search is quite comparable to factorial experiments, since both are sampling uniformly throughout the space (although the factorial approach *enforces* uniformity whereas random search obtains it on average). While we did

not carry out extensive comparisons between different search algorithms' performance, we did compare the genetic algorithm against the baseline of random search (RS), and a hill climbing (HC) algorithm (which is also an evolutionary algorithm in the broader sense of the term, but unlike the GA it is not population-based and it does not use recombination). The hill climber used the same mutation rate as the genetic algorithm, and it was configured to randomly restart (from a new location in the parameter space) after 100 successive failures to make *uphill* progress. The results for the high virality scenario (shown in Figure 5.12) demonstrate that there was a clear (and statistically significant) benefit to using GAs in this case study, compared to both HC and RS. For this task, the hill climber heuristic provided only a marginal benefit over random search, whereas genetic algorithms outperformed both. For the medium virality scenario, performance results (not shown) were very similar. (Further comparisons of search algorithm performance on a variety of model exploration tasks will be covered in Chapter 9.)

5.9. Future Work and Conclusions

In this case study we have only explored one potential adoption heuristic, but a wide range of adoption heuristics exists within the space of contagion/adoption models [Dodds & Watts, 2004]. The Bass-like model investigated here may be the best validated of extant viral marketing models, but it could be useful to look at others, especially since the applicability of adoption heuristics may vary according to product types, e.g., consumer durables vs. software.

In our seeding cost we have assumed that every node costs the same amount to be seeded, but it may be more expensive to seed an individual with more friends. Influentials are influentials because people respect their advice, and thus they are not as easily swayed

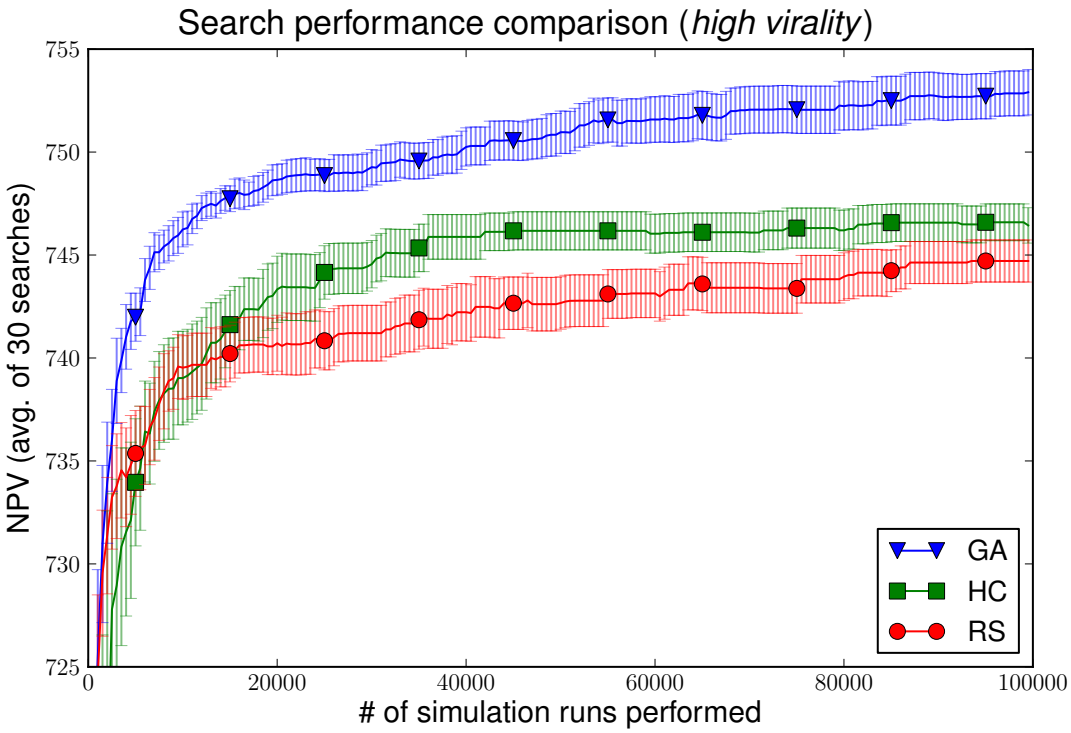


Figure 5.12. Performance comparison for the genetic algorithm (GA), hill climber (HC), and random search (RS) search methods. Error bars show 95% confidence intervals on the mean.

by promotions. The landscape of best possible strategies will alter when different cost functions for seeding are used, especially those that take into account the underlying network characteristics, which are the same features used by the seeding strategies.

While we attempted to choose a representative set of social networks that covered a range of network types, we found that our results were substantially different for our two empirically-based networks (*twitter* and *alumni*) than for the theoretically-based networks. This reminds us of the importance of working with empirical network data in addition to abstract theoretical models. It would be worthwhile to explore alternative network structures, with different degree distributions and different topologies, and most importantly,

other empirical networks should be gathered and examined. Also, it is unclear how well LVMP strategies generalize from a sampled sub-network to the whole network. Future work should include examining how well the strategy derived for our small *twitter* network might apply to the whole Twitter network, or to successively larger subgraphs, to see if the results scale.

In conclusion, we have presented a novel problem (the local viral marketing problem), constructed an agent-based model to simulate consumer behavior for this problem, and showed that evolutionary computation provides a useful method for exploring this space and discovering unexpected features of the problem and the social networks being investigated.

CHAPTER 6

Case Study 3: Artificial Anasazi – Calibration and Sensitivity Analysis

"Digging ... that is the occupation of an archaeologist, my dear."
– Elizabeth Peters, *Lord of the Silent*

"Essentially, all models are wrong, but some are useful."
– George Box

Academic fields of research are often stereotyped: mathematics is about writing proofs, geology is about studying rocks, and archeology is about digging up ancient ruins. While these stereotypes are based on practices which are prevalent in their respective fields, this typecasting of scientists and the methods they use is harmful for two reasons: 1) because outsiders form a constrained view of the rich variety of methods and tools employed within the discipline, and 2) because practitioners of the discipline themselves may be influenced by these cultural stereotypes, and thus limit their own research methodologies to conform (either consciously or subconsciously) to these norms. However, there are many fine examples of iconoclastic research that helps to break down these stereotypes. One of these is the well-known *Artificial Anasazi* simulation, in which a multidisciplinary team of archaeologists, anthropologists, paleoclimatologists, and computer scientists worked together to create a model attempting to explain the rise and fall of a prehistoric society. This simulation provides a "digging-free" method of exploring archaeological research questions.

In this case, the Artificial Anasazi model was able to match the historical record in many respects, but failed to match on one important point: the desertion of the valley. In this sense, the model was "wrong", because it did not reflect reality. However, the model was still "useful" to the researchers, because it showed that the assumptions and simplifications they used when creating the model were insufficient to produce the historical phenomenon, and thus additional factors must have contributed. As George Box noted, all models involve simplifications of the target phenomena, and are thus "wrong" in that sense. This type of "wrongness" is actually part of what *makes models useful*, because if the model contained the full complexity of the target phenomenon, it would provide less insight into which aspects of that phenomenon were most important. However, there are other ways in which models may be "wrong" – such as when there are errors (bugs) in the model's code. In this chapter, we will see an example of how search-based sensitivity analysis helped uncover a bug in a previously published version of the Artificial Anasazi model.

In the previous two case studies, I was personally involved in the development of the models being explored (that is, the Flocking Vee Formations model, and the Diffusion of Product Adoption model). In contrast, this third case study uses an extant model (Artificial Anasazi) that was created by other researchers, and applies the query-based model exploration framework for model analysis. Specifically, we elaborate on the QBME framework presented in Chapter 3 by investigating the use of genetic algorithms (GAs) for performing two common tasks, parameter calibration and sensitivity analysis, which are related to the evaluation of validity and robustness of agent-based models. In the calibration task, we demonstrate that a GA approach is able to find parameters that are equally good or better at minimizing error versus historical data, compared to a previous factorial grid-based approach. The GA approach also allows us to explore a wider range of parameters and

parameter settings. Previous univariate sensitivity analysis on the Artificial Anasazi model did not consider potentially complex/nonlinear interactions between parameters. With the GA-based approach, we perform multivariate sensitivity analysis to discover how greatly the model can diverge from historical data, while the parameters are constrained within a close range of previously calibrated values. We show that by varying multiple parameters within a 10% range, the model can produce dramatically and qualitatively different results, and further demonstrate the utility of sensitivity analysis for model testing, by the discovery of a small coding error. Throughout this case study, we discuss some of the important issues that can arise with calibration and sensitivity analysis of agent-based models.

6.1. Motivation

Agent-based modeling[1] is a technique that is becoming increasingly popular for many scientific endeavors, due to the power it has to simulate complex adaptive systems in a variety of natural and social environments [S. Bankes, 2002; Bryson et al., 2007; Goldstone & Janssen, 2005; Wilensky & Rand, in press]. In an agent-based model (ABM), there are many agents operating according to simple rules, but the resulting interactions between agents lead to the emergence of complex aggregate-level behavior. The resulting aggregate behavior of an ABM (especially one that aims at high fidelity to real-world systems), is often dependent on a large number of controlling parameters. However, because of the complex nature of the emergent patterns, and the nonlinear interactions between these parameters, the outputs of ABMs can rarely be characterized by simple mathematical functions, and formal analytic methods usually prove insufficient [Edmonds & Bryson, 2004]. Furthermore, the computational time required to run an ABM, together with the large number of parameters

[1]Sometimes also referred to as *multi-agent modeling*, *multi-agent based simulation*, or *individual-based modeling*

often makes it infeasible to exhaustively compare all combinations of parameter settings. Additionally, ABMs are predominantly stochastic in nature, leading to variability of results, even when run multiple times with identical simulation parameters. As a result, the rigorous analysis of agent-based models remains a challenging task, and proper methodology for efficient analysis is still at a formative stage. In this work, we offer a case study about the use of one particular approach, genetic algorithms (GAs), to accomplish two common model analysis tasks: parameter calibration, and sensitivity analysis. For this case study, we chose to examine the *Artificial Anasazi* model [Dean et al., 2000].

6.2. Background and Related Work

6.2.1. *Artificial Anasazi* model background

The *Artificial Anasazi* model [Dean et al., 2000; Axtell et al., 2002; Gumerman, Swedlund, Dean, & Epstein, 2003] simulates the rise and fall of the prehistoric Kayenta Anasazi population living in Long House Valley, in northeastern Arizona from the years 800-1350 AD. This agent-based model simulated the residential and agricultural practices of an artificial society at the unit of individual households. It used geographic, rainfall, and various forms of archaeological survey data to achieve a high degree of verisimilitude with respect to historical reality. Moreover, after calibrating their model, the researchers found a reasonably good correspondence between the model and the real history, for both qualitative spatial settlement patterns, and population over time [Axtell et al., 2002].

A particular inspiration for the *Artificial Anasazi* model is to help understand the "fall." Archaeological records demonstrate that the Kayenta Anasazi abandoned the region around 1300 AD. However, the reason for this departure has been debated. One of the primary findings from the *Artificial Anasazi* model is that environmental factors alone were not

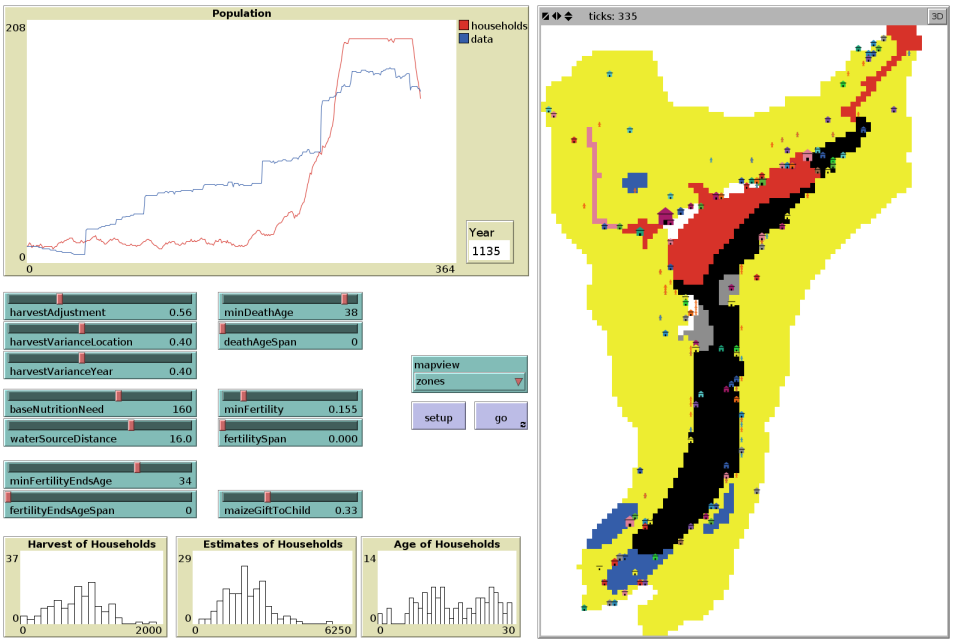


Figure 6.1. Graphical interface of Janssen's NetLogo implementation of the Artificial Anasazi model (with additional model parameters exposed).

sufficient reason for a complete exodus; the valley could have continued to support a modest population [Axtell et al., 2002]. However, for a full discussion of the *Artificial Anasazi* model, we refer the reader to the original sources.

In this work, we analyze the replication of this model[2] by Janssen [2009], which was implemented in the NetLogo modeling environment [Wilensky, 1999]. The model parameters were hard-coded in Janssen's replication, so we converted these variables into "explicit" model parameters that are controllable via the model's graphical interface (see Figure 6.1), as well as making a few minor compatibility changes so the model would run in NetLogo 4.1.[3]

[2]Download available: `http://www.openabm.org/site/model-archive/ArtificialAnasazi`
[3]The exact model file we used is available at: `http://ccl.northwestern.edu/ALIAS/LHV_robustness.nlogo`.

Several reasons motivate our choice of the *Artificial Anasazi* model for this case study. First, whereas many agent-based models are "abstract" models, that demonstrate qualitative trends or emergent phenomena, *Artificial Anasazi* is an example of a "facsimile" model in the typology of agent-based models, which attempts to closely match historical data [G. Gilbert, 2008]. Second, it is a particularly well-known ABM that has received considerable attention, both in the press (e.g., [Kohler, Gumerman, & Reynolds, 2005]), and from the agent-based modeling community in general. Third, there have been several previous calibration efforts using this model [Dean et al., 2000; Janssen, 2009], as well as published (univariate) sensitivity analysis [Janssen, 2009]. We will compare with these prior analyses as we discuss results in the sections below.

Kohler et al. [2000] developed a similar ABM of the nearby Mesa Verde region during this time period, and further elaboration of this model used the *Cultural Algorithm* framework to embed (and evolve) social intelligence within the system [R. Reynolds, Kobti, Kohler, & Yap, 2005; Kobti, Reynolds, & Kohler, 2006]. Despite the commonality of using evolutionary algorithms, our work differs in that we are performing model analysis tasks externally to the model, rather than incorporating evolution as a mechanism within the ABM.

6.2.2. Related methodological research

There are many ways of analyzing and checking for robustness in ABMs, including a variety of approaches for calibrating model parameters and performing sensitivity analysis. [G. Gilbert, 2008; Wilensky & Rand, in press; Chattoe, Saam, & Möhring, 1997]. However, our present work focuses on the use of genetic algorithms for these tasks.

This is not the first time that genetic algorithms have been suggested for parameter calibration and sensitivity analysis of computer simulations. In particular, Miller's [1998]

seminal work on *active nonlinear testing (ANT)* proposed the use of metaheuristic search algorithms for a range of tasks for computer simulations. Specifically, Miller demonstrated how both calibration and a form of multivariate sensitivity analysis could be achieved on the well-known World3 system dynamics (SD) model [Meadows et al., 1974], using either genetic algorithms or a hill-climbing approach. SD models share several features with ABMs, such as nonlinear interactions between parameters. However, SD models tend to model change of aggregate (macro-level) quantities, whereas in ABMs macro-level dynamics emerge from interactions between agents at the micro-level. Additionally, SD models are often deterministic, whereas ABMs are almost always stochastic in nature, requiring the examination of a number of "trials" to evaluate the model's behavior. As we will discuss later, the stochasticity of model run results brings up several important questions about what it means to perform robustness checking on an ABM. Moreover, we believe that the concepts of *active nonlinear testing* deserve further investigation within the context of agent-based models of complex adaptive systems.

Little work has been done in this area, with a few notable exceptions. Calvez and Hutzler [2005] proposed the use of genetic algorithms for tuning the ABM parameters, and demonstrated several parameter tuning tasks on a model of ant foraging [Wilensky, 1997a]. One of these tasks was a mock calibration task, which sought parameters that would yield model output closest to data which had already been generated by the model. In contrast, the *Artificial Anasazi* model represents a *real* calibration task, attempting to match real historical data. Other cases of using genetic algorithms to search the parameter-space of ABMs include: finding optimal seeding strategies for viral marketing in a social network ABM [Stonedahl, Rand, & Wilensky, 2010] (see also Chapter 5), and discovering various forms of emergent collective behavior in flocking ABMs [Stonedahl & Wilensky, 2011] (see

also Chapter 4). We are not aware of previous instances of using genetic algorithms to perform sensitivity analysis on an agent-based model.

6.3. Calibration Task

6.3.1. Task description & prior work

Broadly construed, the *calibration* of an ABM may refer to any process by which changes are made to the model or its model parameters, such that the behavior of the resulting model is closer to a desired behavior. In this chapter, we will more narrowly define *calibration* to be the common case of searching for model parameter settings that yield output that is closest to a specified *reference pattern*. (We will assume that only the model's parameters may be varied, and the model's code is a fixed entity.) In the case of the *Artificial Anasazi* model, we are following two previous calibration efforts [Dean et al., 2000; Janssen, 2009], though we will primarily compare with Janssen [2009] because differences could exist between Janssen's replication and the original model, and also because the original authors' calibration process was not well documented.

Both previous calibration efforts chose the target reference pattern to be the time-series of historical population data (number of households), and sought to minimize an error measure, which defined the "distance" between the simulated population history and the real population history. Following [Dean et al., 2000], we will denote the historical population data with a vector of length 550, X_t^h), where t is the number of years since 800 AD, and similarly denote simulated data with vector X_t^s.

Previous calibration efforts used multiple error measures of the difference between X_t^s and X_t^h, specifically the three L^p norms (L^1, L^2, and L^∞). However, prior work found little difference between the choices of error function, and Janssen [2009] specifically found

that both the L^1 and L^2 measures yielded the exact same optimal calibrated settings[4]. For simplicity our work focuses on the L^2 measure, which is also called the Euclidean distance between the vectors X_t^s and X_t^h. Furthermore, minimizing the L^2 measure yields the same result as minimizing the *mean squared error* when comparing two sequences (the absolute magnitude of the error measures will differ, but finding parameters that minimize $f(x)$ is equivalent to finding parameters that minimize $\sqrt{f(x)}$, for $f(x)$ positive). In terms of the QBME framework discussed in Chapter 3, we are first applying a group-level measure (to get the number of agents present in the model at each time step), and then measuring the *difference* between this information and our reference pattern (at each time step), and then condensing those differences to a single number using the L^2 norm.

Janssen [2009] used a factorial experiment (grid-based sweep) for performing the calibration. Due to computational constraints, Janssen varied only 5 parameters, with 7 to 9 choices for each parameter. In contrast, using a genetic algorithm (or other search-based) approach to calibration makes it feasible to explore a much larger parameter space. Our calibration effort explores a 12-dimensional parameter space, with a wider range of parameter values, and with higher resolution. For a comparison of the parameter calibration ranges we used with the prior calibration effort by Janssen, see Table 6.1. Of course, there is no magic bullet; the model can only be run so many times within a finite time limit. Given a the same amount of computational time, the GA approach can only run the model with the same number of different parameter-settings that the grid-based approach can. However, the GA is a heuristic method that can adaptively explore more advantageous portions of a larger parameter space. The intuition is that by harnessing the biologically-inspired mechanisms

[4]While this may have been true for this particular case, it does not hold in general, as we will see in Chapter 7

Parameter	**Janssen Range** low-high (inc)	**GA Range** low-high (inc)
HarvestAdjustment	0.54-0.7 (0.02)	0.5-1.5 (0.01)
HarvestVarianceLocation	0-0.7 (0.1)*	0-0.5 (0.01)
HarvestVarianceYear	0-0.7 (0.1)*	0-0.5 (0.01)
BaseNutritionNeed	160	100-200 (5)
MinDeathAge	26-40 (2)	26-40 (1)
DeathAgeSpan	0 (const)	0-10 (1)
MinFertilityEndsAge	26-40 (2)	26-40 (1)
FertilityEndsAgeSpan	0 (const)	0-10 (1)
MinFertility	.095-.185 (.015)	0.0-0.2 (0.01)
FertilitySpan	0 (const)	0-0.1 (0.01)
MaizeGiftToChild	0.33 (const)	0-0.5 (0.01)
WaterSourceDistance	16 (const)	6-24 (0.5)

**varied in lock-step, as a single variance parameter*

Table 6.1. Parameter ranges (low, high, and increment) for the GA calibration task, compared with ranges explored in a previous grid-based calibration by Janssen [2009].

of mutation, recombination, and natural selection, the GA will be able to evolve parameter settings that minimize the error measure, and thus calibrate the model. Pragmatically, it is often infeasible to perform calibration with fine resolution on a medium-to-large number of parameters with a grid-based approach. For instance, an exhaustive grid-based search on the parameter space defined for the *GA* in Table 6.1 would involve 6.5×10^{16} combinations of parameters, and would require a million processors each running for over a million years to complete.

6.3.2. Search Method

The GA we employed was a standard generational genetic algorithm [J. Holland, 1975], with a population size of 30, a crossover rate of 0.7, and a mutation rate of 0.05, using tournament selection with tournament size 3.

The value to be assigned to each model parameter was individually encoded in binary using a Gray code.[5] The concatenation of binary sequences for all model parameters forms the genome for an individual in the GA.

Full generational replacement is used, meaning that from each generation of 30 individuals, 30 children are created to replace the parent generation. Each child is created by first using tournament selection to preferentially choose one or two parents with better fitness values, and then performing either sexual or asexual reproduction with the parent(s), followed by per-bit mutation.

To evaluate fitness, the individual is decoded into the component parameter values, the model is run 15 times with those parameters, and fitness function is calculated as the average L^2 error value from these replicate runs. During tournament selection, individuals with lower fitness function values (lower average error) are preferred. The choice to minimize the average 15 replicate runs follows from the previous calibration efforts [Axtell et al., 2002; Janssen, 2009], although we also examine the alternative of searching for the single best run in a second follow-up calibration experiment. In terms of the QBME framework from Chapter 3, this is a choice about how/whether to condense information at the replicate (or repeated model run) level. This choice turns out to have important implications, as we shall see.

To monitor/verify the progress of the GA, for each new "best-so-far" model parameter values that the GA found, an additional 30 independent replicate runs were performed and logged, providing an unbiased (and more confident) estimate of the average L^2 error for those parameter settings. We will refer to this process as *best-checking*, and the verified value as

[5]Gray codes create a smoother mapping between numeric values and binary strings than traditional "high-order" bit encodings, and are thus generally advantageous for search space representations [Caruana & Schaffer, 1988; Whitley, 1999].

the *checked fitness*. (The GA does not make use of *checked fitness* information; rather, this monitoring is extrinsic to the search process.)

Our GA implementation employed *BehaviorSearch*, which is a tool we have developed that interfaces with NetLogo to automate the exploration of ABM parameter-spaces using genetic algorithms or other meta-heuristic search techniques [Stonedahl & Wilensky, 2010a, 2011]. (*BehaviorSearch* will be discussed further in Chapter 10.)

6.3.3. Calibration15 experiment

Using the setup described above, we performed 5 GA searches for parameter settings that yield the best *average of* 15 *model runs*. We will refer to this as the *calibration-15* experiment. Each search went for 100 GA generations, corresponding to running the simulation a total of 45,000 times, with a small number of additional runs used for the extrinsic *best-checking* process. A single GA search required approximately 16.5% of the 272,160 runs required by the factorial-sweep approach employed by Janssen [2009], so the five searches together still required less computation than the grid-base approach. To provide an idea of computational running time, in total these searches required approximately 2500 CPU-hours ($\approx$ 104 CPU-days). Search time is dominated by the time required to run the model and the time spent on genetic operations is inconsequential. Thus, in this chapter we will report computational effort in terms of the number of simulation runs performed.

An examination of search performance of the five *calibration-15* searches shows that one of the five prematurely converged to a suboptimal solution, whereas four of the five reached reasonably good levels of calibration (see Figure 6.2). The best parameter settings found from *calibration-15* experiment (as well as results from later experiments) are given in Table 6.2. These parameter settings yielded a mean L^2 error value of 891.4 (σ = 65.8) from

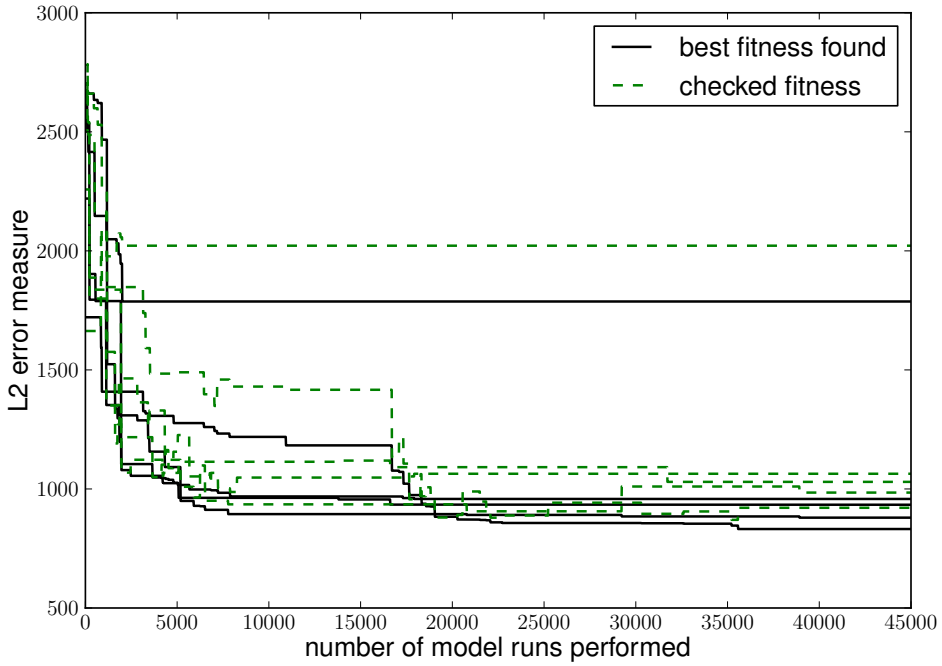


Figure 6.2. GA performance for the *calibration-15* task.

running the model 30 times, which was lower than the mean L^2 error of 945.3 ($\sigma = 80.0$) for the Janssen calibrated settings. Both distributions of error appeared normally distributed (Shapiro-Wilkes test, $p < 0.01$), and the finding that the GA's mean error was less than for the Janssen settings appeared statistically significant (Student's t-test, $p < 0.01$). However, we happened to decide to run the simulation 100 times with each of these settings, and the picture suddenly changed.[6] With 100 replicate runs, the mean L^2 error for the GA parameters was 943.1 ($\sigma = 324.5$), and the mean L^2 error for the Janssen settings was 930.6 ($\sigma = 194.4$); the GA now appeared to have found worse (less calibrated) parameters.

[6]We include this vignette partially as a reminder that statistics must be interpreted with care, and that the distributions of output variables from multi-agent-based simulations may be abnormal, irregular, or generally unexpected.

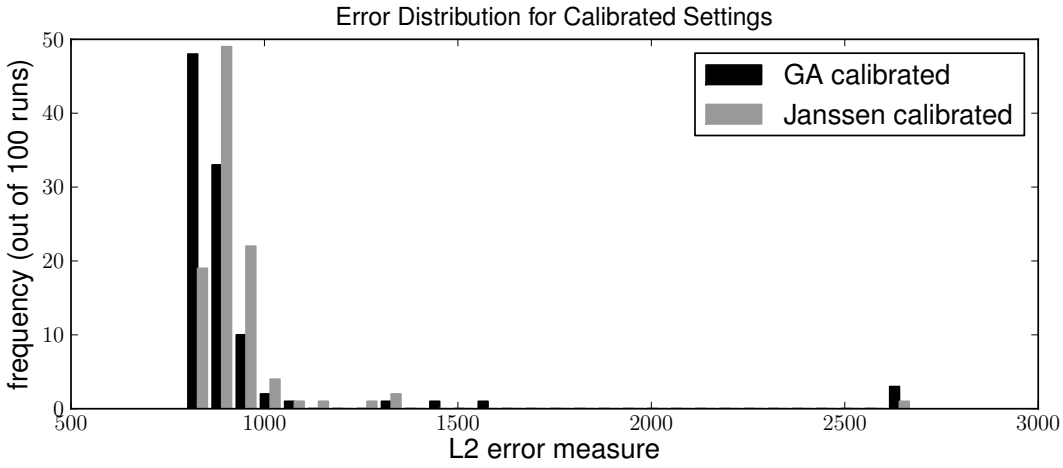
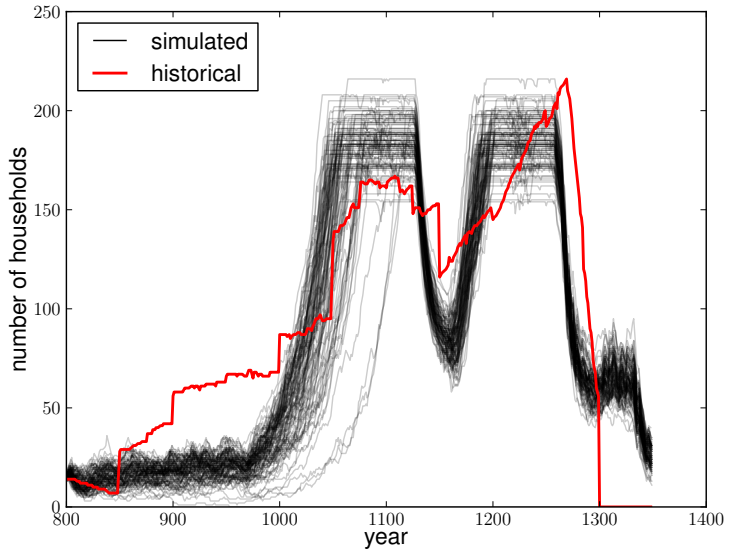
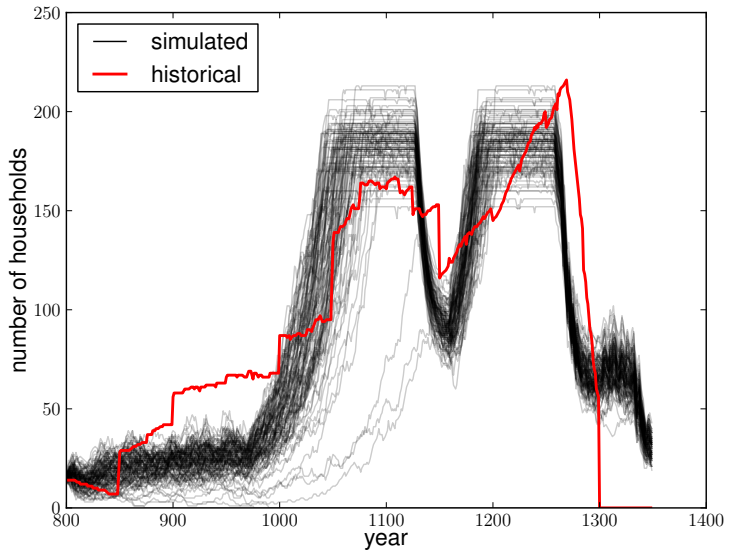


Figure 6.3. A histogram displaying the distribution of error values across multiple runs, comparing the GA calibrated settings with the calibrated settings previously found by Janssen [2009].

This led us to examine the distribution of error among the 100-replicates for each case (see Figure 6.3), which turned out to be non-normal. (In terms of the QBME framework, we are now discussing the impact of *diversity* at the *level* of the fitness function across multiple repeated model runs.) In general, the GA-provided settings usually offer a (slightly) better match with historical data, but there are a few high-error outliers (that raise the mean error value), and these outliers appear more likely with the GA's settings than with Janssen's. These outliers are apparent in the visual comparison of the 100 GA and Janssen simulated histories against the historical data (Figure 6.4). The median L^2 error for the GA was 860.4, compared to 893.8 for the Janssen settings, and a randomly chosen run with the GA settings is almost twice as likely to have better performance than one chosen from the Janssen settings (65.9% vs. 34.1%).



(a) *Janssen calibration*



(b) *GA: calibration-15*

Figure 6.4. Simulated population histories from 100 model runs, showing both Janssen's calibrated settings (a) and the GA's calibrated settings from the *calibration-15* experiment (b), plotted in comparison to the historical data. The flat tops of the simulated trajectories are artifacts of populations reaching simulated carrying capacity, as discussed further in [Janssen, 2009].

The trade-off present here may be described in terms of confidence versus accuracy. Given three hypothetical choices, which of the following represents the *best-calibrated* parameter settings for an ABM?

(1) simulated results are always somewhat close to historical

(2) simulations are often quite close, but occasionally far off

(3) simulated results occasionally match historical data perfectly, but are usually far off

Answering this question is difficult, particularly in facsimile-type models of historical events, since there is only one recorded version of history to compare against (and even for that, the data may be uncertain). We believe that this question warrants explicit consideration whenever a model calibration is performed, and that the choice of distributional comparison may require estimates of the likelihood of history having unfolded in the way that it did, and consideration of plausible alternative histories. For the most part, these estimates and theories will be subjective in nature, which is why it is especially important that they are explicitly addressed during the calibration process. The choice of distributional comparison for calibration will also depend partially on the goals for building the model.

In some cases, one distribution of error may *dominate* another, in the sense that every error value in one distribution is lower than some corresponding error value in the other distribution. In this situation, choosing the "better calibrated" settings is simple, and comparing the mean values is sufficient. However, we would like to emphasize that because of model stochasticity, calibrating ABMs requires comparing one distribution with another, rather than a single result. The issues we have preliminarily touched on here are part of

a potentially much deeper discussion, which is outside the scope of this case study; in future work we plan to formulate a more rigorous and general framework for addressing these aspects of calibration and sensitivity analysis in ABM.

In the case of the Artificial Anasazi model, the GA's distribution of error seems slightly superior to us than Janssen's, given that it usually provides a closer match, and it seems reasonable that in some alternate histories an unlikely adverse chains of events (e.g., poor harvests for many years in succession) could have caused the population's trajectory to be significantly lower (as seen in Figure 6.4). However, the differences in error values are small and one could certainly argue that both the GA's and Janssen's settings are equally well-calibrated; both recreate some features of the historical trajectory while failing to produce others. The fact that the GA was searching a significantly wider range of parameters than Janssen's grid-based approach, yet was not able to find substantially better calibration, suggests that previous calibration efforts on this model were not missing important fruitful areas of the parameter space. However, as the 5 GA searches were only able to cover a small region of the extremely vast search space, this evidence is not necessarily conclusive.

6.3.4. Calibration-1 experiment

The results of the previous experiment led us to wonder how different the results of model calibration would be if we were instead seeking parameters that yielded the single best run, rather than the smallest average error. Investigating this is interesting for several reasons. First, it might discover settings that occasionally match the historical data, even if average error is poor. Second, running the model once is much quicker than running the model 15 times, and although it gives a noisier signal about calibration error, the GA might be able to use this faster noisier fitness function to lead to parameters that provide good

Parameter	**Janssen** calibration	**GA** calibration-15	**GA** calibration-1	**GA** sensitivity-15	**GA** sensitivity-corr
HarvestAdjustment	0.56	0.67	0.64	0.6104	0.5264
HarvestVarianceLocation	0.4	0.47	0.44	0.436	0.436
HarvestVarianceYear	0.4	0.23	0.5	0.424	0.408
BaseNutritionNeed	160	200	185	144	164.8
MinDeathAge	38	37	40	40	41
DeathAgeSpan	0	3	10	1	1
MinFertilityEndsAge	34	36	29	37	31
FertilityEndsAgeSpan	0	9	5	3	0
MinFertility	0.155	0.13	0.17	0.16585	0.14105
FertilitySpan	0	0.09	0.03	0.0155	0.0031
MaizeGiftToChild	0.33	0.31	0.47	0.3102	0.35310
WaterSourceDistance	16	10	11.5	17.44	16

Table 6.2. Optimal parameters found by the genetic algorithm for both the calibration and sensitivity analysis tasks, compared with the parameter settings from the previous grid-based calibration by Janssen [2009].

average performance as well. Because the *calibration-1* experiment requires fewer model runs than the *calibration-15* experiment to evaluate fitness, we were able to increase our genetic algorithm settings to use a population of 90, running for 200 generations, for a total of 18000 simulation runs. We also increased the mutation rate to 3%, as a larger population can generally support a larger mutation rate. Similar to before, we used a *best-checking* routine, this time recording the minimum error from 30 independent replicate runs, each time the GA discovered a new "best." Again we ran 5 searches with these settings, to reduce the risk of reporting anomalous results.

We took the parameter settings corresponding to the lowest *checked fitness* L^2 error (see Table 6.2), and ran the simulation 100 times with those settings. The lowest L^2 error obtained from this was 733.6, which is substantially lower than the 823.5 error that was the best from the 100 runs with Janssen-calibrated settings. These single best runs are compared in Figure 6.5. However, the *average error* for these parameter settings was 962.4,

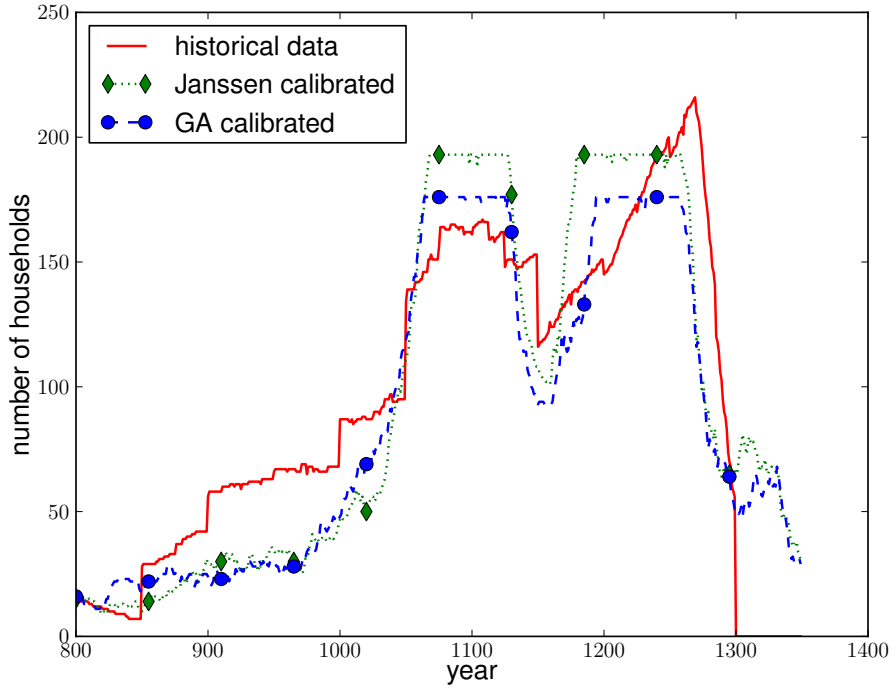


Figure 6.5. The single best runs found from 100 replicate runs with the settings from Janssen (L^2 error = 823.5) and the *calibration-1* experiment (L^2 error = 733.6), compared with historical data.

which is somewhat larger than the mean error for Janssen or *calibration-15*. Essentially, the best *calibration-1* parameters cause more variation in model run results (compare Figure 6.6 with Figure 6.4), which can sometimes lead to a better historical fit, but provides a worse fit if averaged.

This contrast highlights a potential problem with calibrating to get the lowest average error. In order to obtain the absolute lowest average error, every model run would have to be identically equal to the historical data. In general, such a result would indicate a very unrealistic model, where only one path through history is possible. Over the past century, our increased recognition of chaos theory and the effects of path dependence in the social

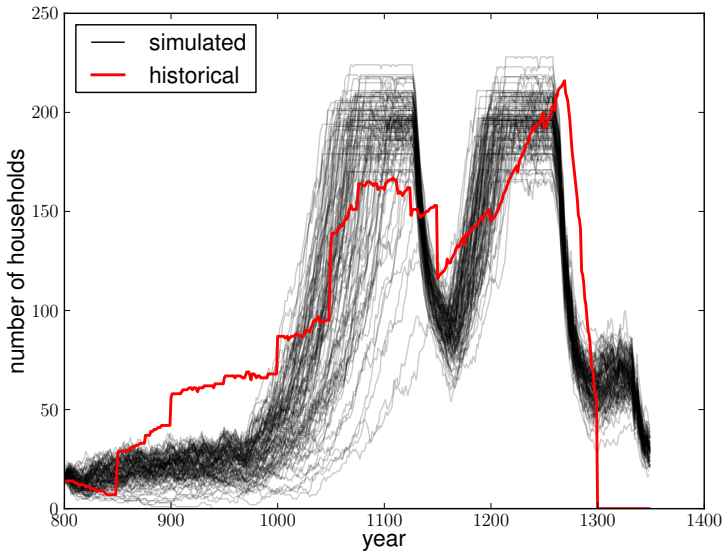


Figure 6.6. Simulated population histories from 100 model runs with the best *calibration-1* parameters, plotted against historical data.

science domain (e.g., [Brown et al., 2005; Batty, 2007]) strongly suggests that small changes in the initial conditions, or chance events early in the process, should significantly influence the historical trajectory. In other words, while a well-calibrated model should be able to produce something resembling the historical data, at least some variation in outcomes is a desirable trait for model credibility. Accordingly, one could argue that the *calibrate-1* experiment provides the best calibrated settings.

6.4. Sensitivity Analysis Task

Sensitivity analysis is a particularly important task, since the robustness (or lack of robustness) of a model with respect to changes in model parameters provides considerable information about the complex system being modeled. However, despite its importance, it is also a practice that is too often neglected by ABM practitioners; if it is performed at all, it often covers only a few parameters, or neglects potentially nonlinear interactions between

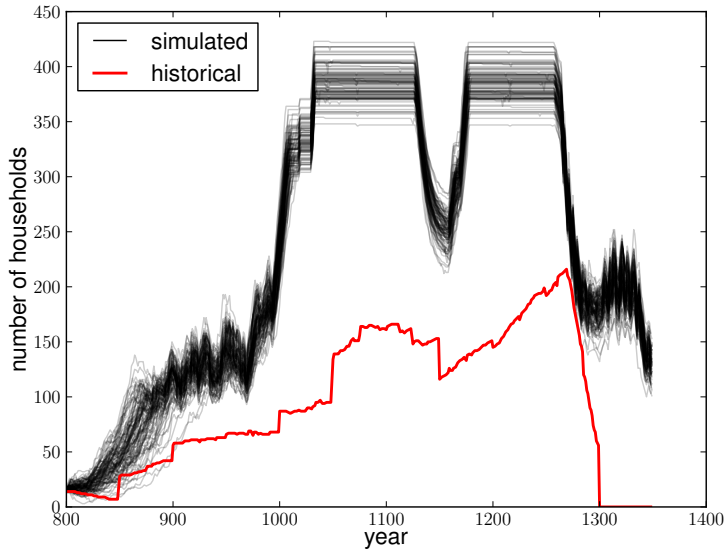
parameters. Some form of sensitivity analysis is a necessary part of ABM verification and validation [G. Gilbert, 2008], as well as replication [Wilensky & Rand, 2007]. However, the term "sensitivity analysis", does not refer to a single precise technique or methodology; rather, the term is broadly applied to class of related techniques that share the goal of determining what factors cause model results to change, and with what magnitude [Chattoe et al., 1997]. In this chapter, we focus only on the specific approach of varying model parameters in the vicinity of some "default" parameter settings. In the case of the Artificial Anasazi model, a partial univariate sensitivity analysis has already been performed. Specifically, Janssen [2009] examined the effect of singly varying each of the five variable parameters from their calibration (HarvestAdjustment, HarvestVariance, MinDeathAge, MinFertilityEndsAge, MinFertility) while holding all other parameters constant (fixed at the previously calibrated values). While this approach does provide insight into the model dynamics near the calibrated point, we are interested in the related question of how robust the model is to changes in multiple parameters simultaneously. Specifically, if model parameters are each constrained to be within a relatively small range of the calibrated values, how far "off" can the model's output be? Exploring this question is one form of multivariate sensitivity analysis, as discussed in Miller's [1998] work on Active Nonlinear Testing. Similar to Janssen's calibration approach, a grid-based factorial parameter-sweep could be employed for small numbers of parameters being swept at low-resolution. However, again we propose an alternative approach of using a genetic algorithm to evolve parameter settings that yield results that are significantly different from the model's desired outcome (i.e. the historical data).

6.4.1. Sensitivity-15 experiment

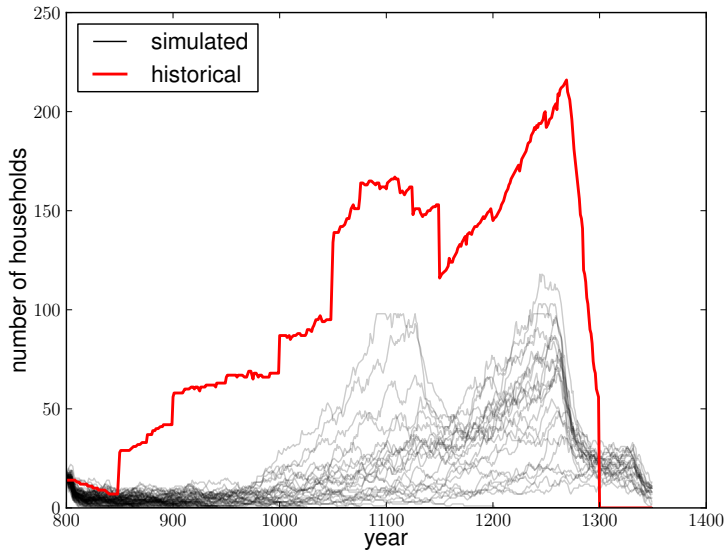
Our first sensitivity analysis experiment was to search for parameter settings, within a small margin of the calibrated settings from Janssen [2009], that would yield the highest average L^2 error measure across 15 runs. Following Miller [1998], we chose to allow each parameter to range within $\pm 10\%$ of its calibrated value. Notice that we only have to change two small things in order to switch from performing model calibration to sensitivity analysis: we restrict the search space to a narrower range for each parameter, and we attempt to maximize (rather than minimize) the same error function (L^2 distance) used for calibration.

Mirroring the *calibrate-15* experiment, we used the same GA settings, and performed 5 searches, each of which ran the model a total of 45000 times[7]. All five of these searches found parameter settings yielding L^2 error values that were more than 4 times greater than the calibrated Janssen settings error (930.6). For the best settings found (again, listed in Table 6.2), the average L^2 error was 3918.6 ($\sigma = 249.7$); Figure 6.7(a) visually displays 100 simulated histories with these settings. While our experiment differs in flavor from that of Janssen [2009], it is still instructive to compare our results with that of the univariate sensitivity analysis previously performed. Specifically, we note that when varying each of 5 parameters singly, the highest relative L^2 error gain was 50% (within the $\pm 10\%$ parameter range), and even the sum of the highest errors for each parameter is only around 150%, which is still small compared with the $> 300\%$ increase in error discovered through the GA's multivariate search. This disparity is due in part to the GA manipulating more parameters to which the model is sensitive (such as BaseNutritionNeed), and also to the nonlinear interactions between parameters.

[7]However, running time in hours was over 80% longer, as these runs tended to create a much greater number of agents



(a) *sensitivity-15*



(b) *sensitivity-corr*

Figure 6.7. Simulated histories from 100 runs with the best sensitivity experiment settings, compared with historical data.

Figure 6.8 displays the distribution of best parameter values found by the GA in each of the 5 searches that cause such a dramatic discrepancy from historical data. (In terms of the QBME framework from Chapter 3, we are now discussing *diversity* at the level of parameter settings that resulted from the search process.) The different GA searches sometimes found different settings from one another, but there are still some clear trends in the results. In particular, they consistently discovered high values for HarvestAdjustment, HarvestVarianceLocation, MinFertility, and MinFertilityEndsAge, while they unanimously selected the lowest possible BaseNutritionNeed value in the range. In other words, the model is particularly sensitive to these parameters. For the most part, these parameter settings match our intuitions. In order to achieve an extremely large population, there should be more bountiful harvests, a higher reproduction rate for creating households, and low nutritional requirements per household. The other parameters' values are relatively scattered throughout the range, and it is apparent that it is not necessary for them to be assigned a specific value in order to achieve large error.

There was, however, a curious trend regarding the two HarvestVarianceX parameters, which raised two questions:

(1) Why does an increase in the variation of crop yield coming from different fields (HarvestVarianceLocation) result in larger populations?

(2) Why is yield variation over time (HarvestVarianceYear) not similarly correlated?

Addressing question 1, we first confirmed this was not a fluke by running the model 100 times with the best *sensitivity-15* settings, except using the lowest HarvestVarianceLocation value in the $\pm 10\%$ range (0.36), and we found a more than 10% decline in L^2 error (t-test, $p < 0.01$). Next, we examined the model code, and discovered that the HarvestVarianceLocation was

affecting agricultural quality as the variance of a normal distribution centered around 1.0, but that agricultural value was not allowed to be negative, so was thus truncated at 0. As a result, increasing the variance also increases the distribution's mean value. The relevant excerpt from the NetLogo model code is as follows:

```
ask patches [
  ;...
  set quality ((random-normal 0 1)
      * harvestVarianceLocation) + 1.0
  if (quality < 0) [set quality 0]
]
```

This explains question 1 from above, and it stems from a reasonable modeling choice, although the outcome shows that one must take care in the interpretation of model parameters. To answer question 2, we looked for where (HarvestVarianceYear) was used in the code, only to find that it wasn't. Instead, HarvestVarianceLocation was *also* affecting variation over time; whereas HarvestVarianceYear was initialized and then never referred to again. This was clearly a bug in the Artificial Anasazi model,[8] which we had uncovered as a result of performing this sensitivity analysis. Admittedly, a careful code audit, or other forms of analysis, could also have helped find this bug. Nonetheless, our GA-based multivariate sensitivity analysis provided the information that led to the discovery of the bug in this published model, which lends further support for the utility of this approach.

From the results, it seems possible that it would be sufficient to only test the extreme settings (+10%, and −10%), rather than checking all values in between. With 12 parameters,

[8]We reported this issue in personal correspondence with the model author. We also note that this minor error did not affect any of the results previously obtained in [Janssen, 2009].

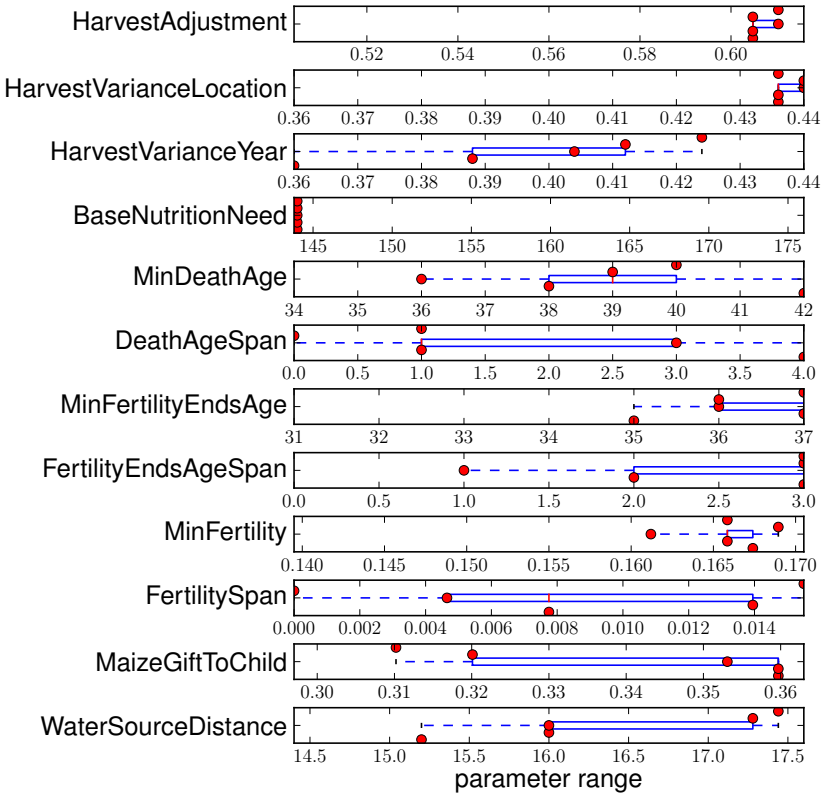


Figure 6.8. Distribution of "best" parameter settings found in each of the 5 GA searches of the *sensitivity-15* experiment. Actual parameter values are displayed as solid circles, while the boxes and whiskers display the middle 3 runs, and full extent of the data, respectively. The center x-value in each plot corresponds to the Janssen calibrated settings.

this would only require $2^{12} = 4196$ combinations of parameter settings, which is a feasible number to enumerate. This may often be the case, but in general one cannot be sure that nonlinear interactions between parameters would not cause the optimal/extreme results to fall elsewhere in the viable range. For models with very large numbers of parameters, and small viable ranges for each parameter, allowing only 2 or 3 choices for each parameter may be prudent, together with a genetic algorithm approach.

We also performed a *sensitivity-1* experiment, using similar settings as the *calibration-1* experiment, searching for parameters that would cause the largest L^2 error for a single model run. However, the results were very similar to the *sensitivity-15* experiment, and are thus omitted for the sake of brevity.

6.4.2. Sensitivity-corr experiment

Although the *sensitivity-15* experiment produced results of a different quantitative magnitude than results from calibrated values, they were still qualitatively similar (see Figure 6.7(a)). We were interested in whether we could use a different error measure for a sensitivity analysis to find simulated histories with a different general shape. As a measure for qualitative difference, we chose the Pearson product-moment correlation coefficient (r) between the simulated (X_t^s) and historical (X_t^h) population sequences. As an example, the single run with the largest L^2 error value (4524.3) from the *sensitivity-15* experiment still had a quite high positive correlation ($r = 0.83$) compared with the historical data.

Using a genetic algorithm with the same settings as the *sensitivity-1* experiment (population 90, 200 generations, 3% mutation), we ran 5 searches for parameters (within the $\pm 10\%$ range) that would yield the smallest correlation coefficient (r) value. The best (lowest correlation) parameter settings are listed in Table 6.2, yielding an average correlation of $r = -0.18$. Whereas the largest L^2 error measure was achieved by an unrealistically large Anasazi population, the smallest correlation was achieved by population decline and extinction, which are also consistently achievable within the $\pm 10\%$ range of calibrated values. Of 100 runs (shown in Figure 6.7(b)) using the best parameters for non-correlation, the lowest correlation for a single run was -0.6, which had a relatively long lingering decline with the population reaching 0 in the year 994 AD. Interestingly, because of our chosen measure, slow

population declines cause greater negative correlation with the data than when the population dies out almost immediately. This led the GA to find runs that were on the brink of extinction, and thus out of the 100 runs, there are a few runs that are still highly correlated with the historical data (the closest matches in 6.7(b)). Though the Pearson correlation-coefficient was reasonably effective in this case for finding qualitatively different runs, it is worth emphasizing that it may not always be appropriate. Developing a variety of error measures for search-based sensitivity analysis that correspond well with human intuitions about what constitutes qualitatively different behavior of a system is a ripe area for future work.

6.5. Conclusions

To summarize, we have presented a series of 5 experiments using genetic algorithms to perform tasks relating to ABM calibration and sensitivity. In the calibration tasks, we demonstrated that the genetic algorithm could find calibrated parameters that were better (in some respects) than parameters previously discovered in a grid-based sweep. This process brought up important aspects of calibration (judging distributions of error, rather than simply mean error), which researchers should attend to during model analysis. In the sensitivity tasks, we demonstrated that the genetic algorithm approach can consistently find parameter settings that yield both dramatically and qualitatively different results. Additionally, the multivariate sensitivity analysis highlighted several instances of anomalous model behavior, leading us to discover a bug in the Artificial Anasazi model's code. This emphasizes the utility of sensitivity analysis as a technique for model testing and verification. Several of the issues about search-based robustness-checking that arose from this case study deserve further consideration, some of which will be discussed in later chapters. For instance, Chapter

7 will dig a little deeper into the trade-offs between using different error measures for calibration as the fitness function for genetic algorithms, Chapter 8 will investigate one aspect of how model stochasticity (leading to noisy fitness functions) impacts search performance, and Chapter 9 will provide performance comparisons for the effectiveness of different search methods. However, some important outstanding questions remain: how should comparisons best be made across spatial and temporal data?[9] what is the most appropriate method for comparing distributional outcomes against single-instances? how should models be calibrated using higher-dimensional or network-based data sets? A complete methodological framework for addressing these questions is outside the scope of this document, but it is an important area for future work in the calibration and sensitivity analysis of ABMs. However, the results of the present study of the Artificial Anasazi model are both thought-provoking and promising, and it is our hope that ABM practitioners will adopt similar methods to improve the rigor of model analysis.

[9]However, there has been some recent progress in this area – see, e.g., [Brown et al., 2005].

CHAPTER 7

Case Study 4: Online News Consumption – Calibration Comparison

"Today's scientists have substituted mathematics for experiments, and they wander off through equation after equation, and eventually build a structure which has no relation to reality."
– Nikola Tesla (1934)

"Not all those who wander are lost."
– J.R.R. Tolkien, *The Fellowship of the Ring*

The paired quotations above are illustrative of a certain yin yang relationship that exists in scientific modeling – a balance between the elegance and simplicity of theory and the practicality of empirical data and experimental evidence. It is important to ground modeling research with real-world data, while still retaining the freedom to "wander" through various simulation microworlds that one may construct that are simpler or more elegant, to gain insight through exploration of isolated aspects of the phenomena. Fortunately, agent-based modeling does have some advantages in this regard over the equation-based modeling paradigm that Tesla maligned. This is because one can often map more directly between the agents being modeled and the real-world entities they represent, whereas in mathematics the equations can quickly become very abstract and difficult to interpret in the target domain.

These quotations also seemed apropos for second reason because this chapter involves modeling *wandering* itself – specifically, modeling the behavior of consumers of online content as they wander (lost or not) from one website to another, reading news stories. However, the specific topic of the case study is less important than the bigger picture, which is about investigating the relationship between calibration measures and their corresponding fitness functions when employed by a genetic algorithm to search the parameter space. Agent-based models can be manipulated to replicate real-world patterns, but finding parameters that achieve the best match can be difficult. To validate the model, the real-world dataset is often divided into a training set (to calibrate the parameters) and a test set (to validate the calibrated model). The difference between the training and test data and the simulated data is determined using an error measure. In the context of using an evolutionary computation technique to calibrate model parameters, the error measure also serves as a fitness function, and thus affects evolutionary search dynamics. This chapter surveys the effect of five different error measures on both a toy problem and a real world problem, using an agent-based model to match online news consumption behavior. We use each error measure separately for calibration on the training dataset, and then examine the results of all five error measures on both the training and testing datasets. We show that sometimes certain error measures serve as better fitness functions than others, and in fact searching for one measure may result in better calibration (on a different measure) than searching for that measure directly. For the toy problem, Pearson's correlation measure dominated all other measures, but for the more complex real-world problem no single error measure was Pareto dominant.

7.1. Motivation

Agent-based models (ABMs) and other modern computational simulations tend to produce a large quantity of data, which is often longitudinal in nature. Moreover, in order to properly utilize these models, since they are stochastic, it is often necessary to replicate runs of the same parameter settings to create multiple datasets so that the statistical variance present in the stochastic nature of the model can be captured [North & Macal, 2007]. Often the goal is to show that these models can simulate real world behavior, a process known as *validation* [Conway, Johnson, & Maxwell, 1959]. However, in order to match model data set, M, against real world data set, R, there is often a large space of parameters, P, that needs to be calibrated so that the simulated data best matches the real data, but choosing the set of parameters that will maximize this match can be difficult. In order to identify the best set of parameters, the real-world data set is often divided into two subsets: (1) the training set R_{train}, and (2) the test set R_{test}. Ideally, R_{train} and R_{test} will both be equally representative of the phenomena being modeled and collected under similar real-world conditions. However, these two datasets may vary in a number of different ways, such as the size of the dataset, the environment that they were collected in, etc. Thus, we must specify an additional set of environmental variables, E, for each scenario: E_{train} and E_{test}. The problem of calibration can now be posed as a straightforward search problem: Identify the set of parameters P^* such that some error measure $\epsilon(R_{train}, M(P^*, E_{train}))$ is minimized. Once the model has been calibrated using P^*, the results can be validated by comparing the model data to the test set using, $\epsilon(R_{test}, M(P^*, E_{test}))$, if this result is less than some threshold, T, then the model is said to be validated.

Many techniques can be used to search for the parameter set, P^*, but if the problem contains many different variables that are interdependent, then an evolutionary computation approach is often suitable. In order to conduct this search, a population of potential parameter sets is generated and the fitness of each individual, P_i, is measured using the error measure, $\epsilon(R_{train}, M(P_i, E_{train})$. In the context of an evolutionary algorithm, the error measure, ϵ, becomes the fitness function, and so choosing the appropriate error measure is critical not only to choosing a good set of parameters, but also to the evolutionary process. As is the case of many evolutionary computation problems, the question then becomes which fitness functions to choose [D. E. Goldberg, 1989; Ma & Abdulhai, 2002]? The choice of the fitness function is important for two different reasons: (1) it will affect the performance of the evolutionary algorithm, and (2) because it is the basis for calibrating the model and judging the validity of the model.

To investigate the effect of an error measure on calibration and validation, we examine a variety of error measures in the context of a real-world problem concerning online news consumption. Specifically, we seek to discover the extent to which the underlying hyperlink network between news sites can explain individual consumer browsing behavior, ignoring content-specific issues and focusing solely on structural network properties and positions of websites in the network. Such a model could be used to investigate how changes in online news business models would affect consumption, e.g., whether the current push to implementing paywalls[1] around news sites will dramatically affect network traffic.

We begin this chapter by briefly describing related work on the calibration of agent-based models and online news consumption. We will then discuss the news consumption

[1] "Why the NYT Will Lose to HuffPo", Felix Salmon, *Reuters*, Feb. 8, 2011. `http://blogs.reuters.com/felix-salmon/2011/02/08/why-the-nyt-will-lose-to-huffpo/`

data that we analyzed, the model that we built and a "toy problem" which we used in our analysis. Next, we discuss five candidate calibration/error measures and the general calibration procedure, followed by a description of our data set, which consists of clickstream data from thousands of individuals consuming news on the Internet. After this, we will go on to discuss the implementation of our agent-based model (ABM) of consumer behavior. We first examine the results of using different error measures on the toy problem, and then on the real world dataset, and summarize our findings. Finally, we examine the use of different error measures using the real-world dataset (for both training and testing), and summarize our findings.

7.2. Related Work

Agent-based modeling is an increasingly popular form of computer simulation, wherein a set of behavioral rules are specified at the individual level, the execution of which results in trends emerging at the system/aggregate-level [North & Macal, 2007; S. Bankes, 2002; N. Gilbert & Troitzsch, 2005; Wilensky & Rand, in press]. Along with other simulation techniques, it often requires the specification of a large number of parameters that affect both individual behavior and environmental factors in the simulation, and machine learning approaches, such as genetic algorithms (GAs) are often brought to bear in these circumstances. While GAs [J. Holland, 1975; D. E. Goldberg, 1989] have long been used to explore computer simulation parameters (e.g., [Weinberg, 1970]), there is an increasing amount of research using GAs in conjunction with agent-based models. For example, Midgley et al. [2007] used a GA to explore an ABM of a consumer retail environment, and Stonedahl et al. [2010] (see also Chapter 5) demonstrated the use of GAs for searching for ABM parameters

in the context of discovering good viral marketing strategies. Specifically regarding calibration using GAs, Miller's [1998] seminal work on "active nonlinear testing" (ANT) proposed the use of nonlinear optimization techniques (including GAs) for a number of important model analysis tasks, including calibration. Miller [1998] demonstrated these ideas using a deterministic equation-based systems dynamics model. In contrast, here we are attempting to calibrate a stochastic agent-based model that we have developed, and more importantly, are investigating the use of a variety of calibration measures and their impact on the GA's performance. Calvez and Hutzler [2005] used a GA for an artificially constructed calibration task in an ant colony foraging ABM, attempting to match previously simulated data, using Euclidean distance (L^2 norm) to measure error. In the preceding chapter (Chapter 6), we demonstrated the effectiveness of GAs for calibrating and analyzing the parameters of the Artificial Anasazi model. That work focused solely on the L^2 norm for calibration, for the sake of matching previous grid-based (factorial) calibration experiments [Janssen, 2009] on that same model, which showed little difference between the L^1, L^2, and L^∞ error measures. In this chapter, we show that the choice of error measure can make a difference in the both the parameter settings (P^*) that result, as well as the GA's performance.

The model we are calibrating is in the application area of online news consumption. There have been several surveys of how people consume news [Althaus & Tewksbury, 2000; Dutta-Bergman, 2006] such as the Pew Internet & American Life Project's recent report [Purcell, Rainie, Mitchell, Rosenstiel, & Olmstead, 2010], but these surveys describe *stated preferences* and not *revealed preferences*. Also, the surveys generally do not provide prescriptive guidelines about how users might react to changes in the content news world. Tewksbury [2003; 2005] examined both survey and URL data, but was primarily focused on what topics people choose to read about and not how they consume news. Our work fills this gap by

empirically examining not just what consumers read, but how consumers browse, and the connection between browsing behavior and the underlying hyperlink network.

7.3. News Consumption

A brief description of the purpose of the model will help motivate its use as an example. Before the growth of the internet, newspapers, essentially, had a geographic monopoly on the area that they served. However, with the development of the web and hyperlinked structures of content, every newspaper had to compete with every other newspaper in existence. As a result, they had to develop new revenue models, such as paywalls, public-sponsored journalism, or consortiums of independent journalists, in order to deliver the same level of quality they were able to deliver in the past. Unfortunately these new models are not based on rigorous models of consumer behavior. Before a newspaper can understand the implications of these new revenue models it is first necessary to understand how users consume news online so that projections can be made as to the effect of different revenue models.

Besides being a highly topical and relevant research area, this domain also has benefits for investigating the effect of calibration measures because there is a large quantity of real-world data, it is embedded in a distributed network, and the question revolves around finding the parameters of an individual-level model that will produce emergent-level outputs that resemble the real-world patterns. Additionally, the large amount of temporal data available allows us to calibrate the model in one time period and then test the results on a separate dataset.

7.3.1. The Data

The data used for this analysis was clickstream data from comScore. This dataset contains approximately 2 million page views per month, from a random sample of a thousand internet users during the year 2007. We divided this data into two smaller datasets: one containing only January browsing (for training) and one containing only December browsing (for testing). Among other information, these datasets contain the referral domain and the destination domain for every link clicked by each of the tracked users. For each of the two datasets, we created a weighted directed graph where each node (vertex) represents a web domain (e.g., nytimes.com), and a directed edge was placed between any node A and node B if there were any hyperlinks clicked to travel from domain A to domain B. Each edge was also assigned a weight, based on the amount of traffic (number of hyperlinks clicked) from one site to another.

However, since our focus is on news consumption, rather than web browsing in general, and because modeling the whole web is infeasible, we further filtered the dataset based on a list of 455 domains identified as news top websites. Specifically, we included any site that was in the top 100 news category from Alexa traffic rankings, combined with a list created by Hasan et al. [2010] of the top news websites and blogs from the time period. We kept only those edges for which the source and destination nodes were both in the list of news sites. This reduced the size of the networks from over 80,000 nodes to 422 nodes and 3113 edges (for January) and 417 nodes and 3086 edges (for December). Since we were primarily concerned with cross-site browsing, we ignored intra-site links (which corresponds to excluding self-loops from our graph representation). We also recorded the total amount

of incoming traffic for each node that arrived either directly (e.g, via bookmark, clicked link from email, etc.), or via some website not in our list of news sites.

Based on preliminary analysis of empirical data, we were able to show that network position has an impact on how much traffic a website receives, and how often it is used as a starting point for browsing. We found that, regardless of the website's size (measured by the traffic it receives in a month), the more central a node is, and the more it will be used as an anchor node. A preliminary regression model on the data showed that highly central nodes gain traffic over time, while less central, and more clustered nodes will lose traffic, even when the original node size is controlled for.

Building upon these results we constructed a simulation that models the traffic across the network. The training data (R_{train}) that we will be trying to match with our agent-based model (described in section 7.3.2) consists of the quantity of traffic on each of the edges during January, with the environment (E_{train}) consisting of the unweighted version of the graph (which is a proxy for the hyperlink structure) and the probabilities of entering the graph (from the outside world) at each node. Similarly, R_{test} and E_{test} are composed of the equivalent data for the month of December. Things move quickly in Internet time: the December network and traffic is substantially different from in January, with only around 60% of traffic volume remaining on the same links. A visualization of the January graph is shown in Figure 7.1, illustrating a dense cluster of sites in the center, with many peripheral (mostly low-traffic) sites surrounding it.

7.3.2. Model Implementation

Using the NetLogo [Wilensky, 1999] multi-agent modeling language, we developed a simple agent-based model of consumer browsing behavior, premised on the idea that a consumer's

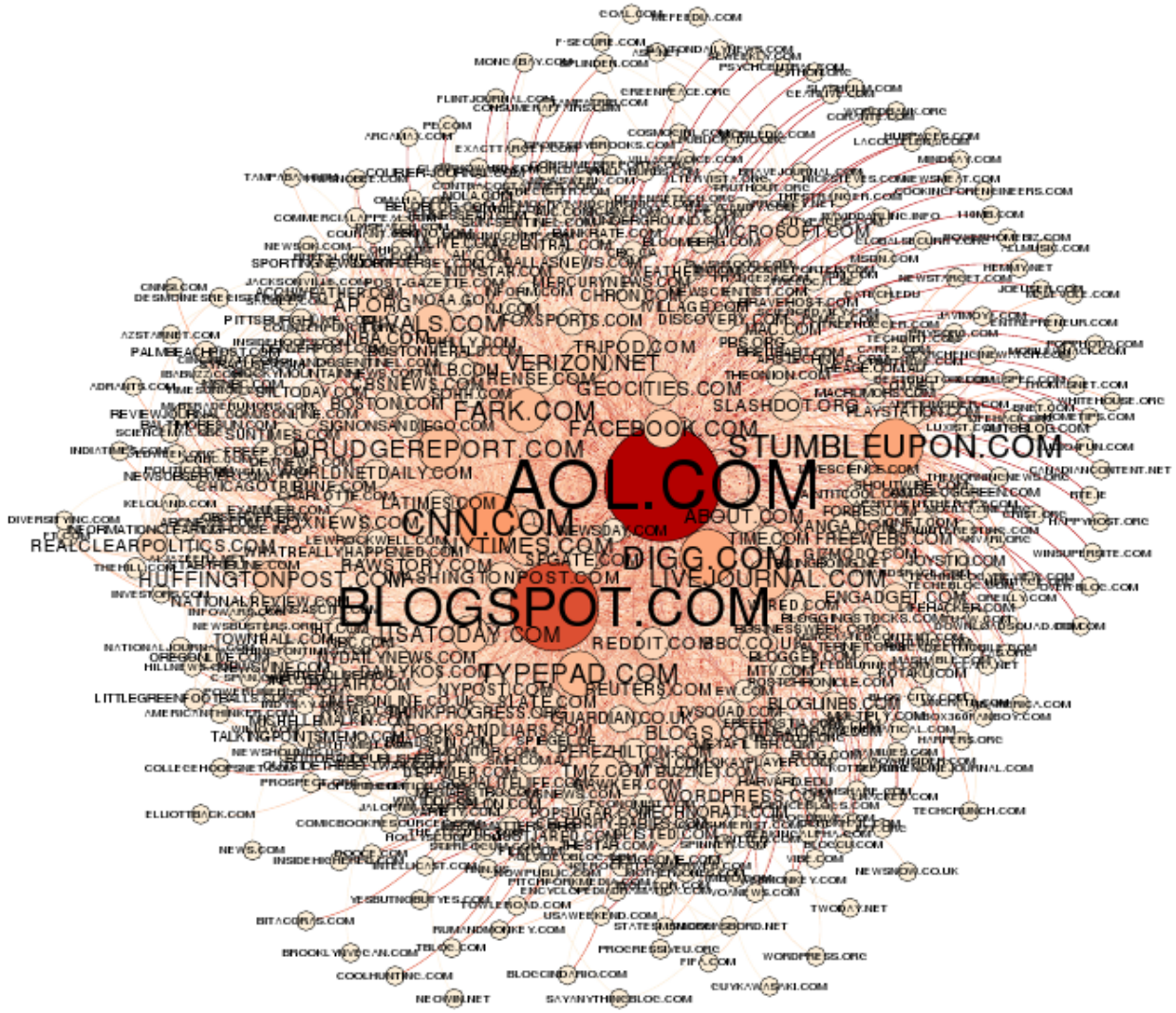


Figure 7.1. Visualization of the directed link network for the January comScore dataset. Node size/color both reflect the total number of observed incoming and outgoing hyperlinks for each website.

decision of which link to click on may be approximated as a function of the structure of the observable inter-site hyperlink network. That is, given an *unweighted* directed network, which merely shows the possible links to other news websites, an agent may choose among the link options based on network-theoretic properties of the candidate destination nodes.

Of course, it is ludicrous to imagine that consumers actually compute network measures for each candidate website, when they are considering whether to follow a link to site A or site B. Rather, these measures should be taken to characterize (or be correlated with) unobserved properties of the site - e.g., a site with a high betweenness centrality is one that serves a connecting role to the news website world, whereas one with a high number of out-links to other sites is likely to be more of a news aggregator, and one with a high in-degree or PageRank might be interpreted as being a producer. We also acknowledge that this model ignores one of the most obvious factors that people (consciously) use when consuming web-based news stories – namely, the title and/or content of the news article the link is pointing to, i.e., the implied quality of the content. However, a key purpose of building this model was to find how much leverage we can get out of the network structure, in terms of predicting consumer traffic, while ignoring the content.

To explore this hypothesis, agents are given a ranking function, $f(N)$, which is parameterized by weighting coefficients corresponding to the relative importance of a variety of structural node-level network statistics. This ranking function takes a candidate node N as input, and produces a real-valued score representing the appeal of moving to that node. The ranking function is a linear combination of weighting coefficients corresponding to each of the following properties:

- randomness - random value injecting noise into the ranking, allowing agents to choose links stochastically.
- in-degree - # of incoming links to this node
- out-degree - # of outgoing links from this node
- in-component - # of nodes that can reach this node
- out-component - # of nodes reachable from this node

- pagerank - PageRank score (with $\alpha = 0.85$) [Langville & Meyer, 2005]
- hits-hubs - HITS hub score for this node [Langville & Meyer, 2005]
- hits-authorities - HITS authority score for this node [Langville & Meyer, 2005]
- clustering - clustering coefficient for this node (calculated on the undirected version of the graph)
- betweenness - betweenness centrality of the given node
- eigen - eigenvector centrality of the given node

Each of the weighting coefficients is a model parameter which can vary between -1.0 (biased against) and 1.0 (biased for), except for the special randomness weight, which varies between 0.0 and 2.0. For example, assuming all other weights were 0, an in-degree weight of 0.8 and a clustering weight of -0.4 would correspond to a movement rule that prefers following links to nodes that have a large number of in-bound links and a low clustering coefficient. Since the randomness weight is 0, this movement rule would deterministically choose the same path through the network, given the same starting point. As a second example, if the betweenness weight were 0.5 and the randomness weight were 0.5, the movement strategy would prefer moving to nodes that have high betweenness centrality, but would also give equal weight to randomness in its decisions.

Besides the ranking function described above, we include two additional parameters to control the behavior of the agent: no-backtrack, which prevents an agent from going back to a node they just visited, and random-restart, which controls how often the agent starts a new browsing session. Given the ranking function and these parameters, the web surfing agent's behavior is as follows.

(1) The agent starts at a random node, chosen with probability proportional to the empirically observed likelihood of someone arriving at that node (either from a non-news website, a bookmark, etc).

(2) The agent forms a set of candidates from all of the nodes that are reachable by outgoing links from its current location. If no-backtrack is set to TRUE, then the node that the agent just traveled from (if any) is excluded from the set.

(3) If the candidate set is empty, or if a random variable is less than random-restart, the agent restarts at a new location, i.e., go to step 1.

(4) Otherwise, the agent computes the appeal of following each link by computing the ranking function across the candidate nodes. To guarantee that the network characteristics are each being given equal weight, the node-level characteristics of each of the candidate nodes are normalized by dividing by the sum of that characteristic across all candidate nodes.

(5) The agent then chooses the candidate with the highest ranking function score. The agent follows the link to that node, and we record a "click" on the link.

(6) Until some specified number of clicks have occurred, go to step 2 and repeat.

The output of the model (M) is the simulated traffic distribution (how many times each link was followed). When the model is run on an empirical network, this output can then be compared with real world traffic data, and we can attempt to calibrate the 13 model parameters to improve the match (as described in Section 7.4). In terms of the QBME framework introduced in Chapter 3, unlike in the Artificial Anasazi model, we are not condensing information across time - instead, we are only using the final slice of data in the temporal dimension, and discarding all history about how it reached that state. Furthermore, there are three types of agents in this model (websites, hyperlinks, and web

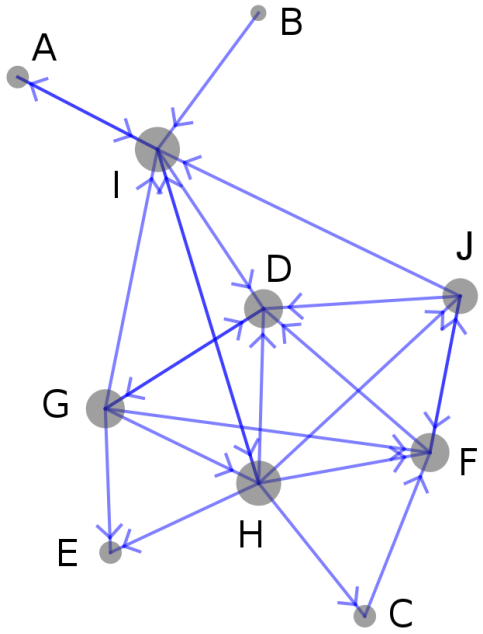


Figure 7.2. The directed network for the *toy problem*.

surfers), and we are only extracting information from the hyperlinks to compare it with a reference pattern. Because this basic version of the model does not include interaction between agents and because an agent restarting is equivalent to a new agent entering the system, we ran the simulation with just a single agent. In its current form, the surfing agents are also memoryless (or a one-step memory, if no-backtrack $= true$), making the simulation interpretable as approximating the steady-state distribution of a stochastic Markov process on the hyperlink graph structure.

Since the actual network is very complex and since we do not actually know the underlying rules that consumers use when moving between nodes in a network, we also created a "toy" network and dataset so that we can explore the effect of calibration measures in a world where the ground truth is actually known. We generated a small random graph of 10 nodes and 23 links (see Figure 7.2). We then initiated the model using a set of parameters that

were similar to parameters discovered in the real world dataset based on early runs of the GA: random-restart $= 0.15$, randomness $= 0.40$, out-degree $= -0.1$, in-component $= 0.2$, out-component $= 0.2$, eigen $= 0.1$, no-trackback $= false$, and all other parameters set to 0.0. We also assumed that the likelihood of starting at any of the nodes was equal. We then ran this model once to generate a ground truth data set similar to the real world news consumption data set.

7.4. Calibration

Regardless of whether we are examining the toy problem or the real world problem, in order to match the model data, M, against real world data, R, we must first divide R into R_{train} and R_{test}, with corresponding environmental variables E_{train} and E_{test}. Calibration is then accomplished by identifying the set of parameters P^* such that an error measure $\epsilon(R_{train}, M(P^*, E_{train}))$ is minimized, and the model can be validated by examining $\epsilon(R_{test}, M(P^*, E_{test}))$. In our case we use a GA with the error function, ϵ, as the fitness function, and each individual being a potential parameter set, P. In this next section we define five different error functions, and the specific calibration that we carried out.

7.4.1. Calibration measures

For this work, we assume that the real world data R and the model's output M can both be represented using fixed-length vectors of numeric values (V_R and V_M, respectively). Since it is impossible to make a comprehensive list and test all possible error measures, we chose to investigate a set of five specific error measures for model calibration: correlation, and four different measures of the L^p norms. These five commonly-used measures span a range of

what they emphasize, but they all capture some notion of distance between empirical and model data.

(1) *corr* - Pearson's product-moment correlation coefficient[2]. Since correlation values range between -1 (perfectly anti-correlated) through 0 (uncorrelated) to 1 (perfectly correlated), the "error function" we are technically minimizing here is the negative of the correlation. However, for simplicity of interpretation, we will report all calibration values as the correlation (Pearson's r). As a result, for this measure only, a higher value will indicate a closer match to the empirical data.

$$corr = \frac{\sum_{i=0}^{n-1}(V_M(i) - \bar{V_M})(V_R(i) - \bar{V_R})}{(n-1)\sigma_{V_M}\sigma_{V_R}} \tag{7.1}$$

where σ_{V_M} σ_{V_R} are the standard deviations of V_M and V_R respectively.

(2) L^0 - This is an extension of the L^p norms to the case where $p = 0$. $L^0(V_M, V_R) =$ (the number of positions where the two vectors differ). Note that the magnitude by which they differ does not matter.

(3) L^1 - The L^1 norm, commonly known as "Manhattan distance", is computed by:

$$L^1 = \sum_{i=0}^{n-1} |V_M(i) - V_R(i)| \tag{7.2}$$

(4) L^2 - The L^2 norm, commonly known as "Euclidean distance", is computed by:

$$L^2 = \sqrt{\sum_{i=0}^{n-1} |V_M(i) - V_R(i)|^2} \tag{7.3}$$

[2]Stonedahl and Wilensky [2010b] also proposed using the correlation coefficient as a fitness function, though not in the context of calibration, but rather for the converse task of sensitivity analysis and model testing.

(5) L^∞ - The L^∞ norm, also known as "Chebyshev distance" or "maximum metric", is computed by:

$$L^\infty = \max_i |V_M(i) - V_R(i)| \tag{7.4}$$

The four L^p norms comprise a spectrum of calibration choices: L^0 ignores magnitude and cares only about the quantity of errors, while L^∞ ignores quantity and cares only about the magnitude of the largest error. The L^1 and L^2 measures fall in between. Also, note that minimizing the L^1 norm is the same as minimizing mean absolute error, and the L^2 norm is equivalent to minimizing either mean square error (MSE) or root mean square error (RMSE).

The *corr* correlation function belongs to a different family of error measures. However, the *corr* function has several interesting properties, including that it is invariant to both location and scale. That is, perfect correlation can be achieved when $V_R = \alpha V_M + \beta$ where α and β are scalar constants. Whether this is desirable depends on your situation and calibration goals. In our case of attempting to match hyperlink traffic between news sites, if our model can produce numbers that are correctly correlated with the traffic on each link, then we would consider that a successful calibration. Scale invariance has the benefit of needing shorter simulations using fewer agents to compare to real data using large numbers of people over long time periods. However, in the current work, in order to successfully examine all of the proposed error messages, we run our simulation until it has created exactly the same amount of hyperlink traffic as the real-world dataset, so it is theoretically possible for all calibration measures to attain a perfect match.

7.4.2. Calibration method

To perform the actual calibration, we used *BehaviorSearch*[Stonedahl & Wilensky, 2010a] (see also Chapter 10), which provides facilities for exploring the parameter space of agent-based simulations using GAs. Specifically, within BehaviorSearch we used a steady-state GA with population 50, tournament selection (tournament size 3), and replacement strategy of replacing a random individual in the population. We used a crossover rate of 70%, with a mutation-chance of 5% per locus, and one-point crossover.

The only complicated part of the GA setup was the genotype encoding, which used a hybrid real-valued and boolean chromosomal representation, and included some engineered epistatic interactions between genes. The random-restart parameter was real-coded, and the no-backtrack parameter was boolean. The 11 ranking coefficients were real-coded, but after each coefficient gene we inserted a boolean gene that controlled whether the previous coefficient was expressed. The intuition behind this arrangement was that we had given the model a fairly large number of potential network-statistics to include in the ranking function, and it was unknown which would be useful. Providing epistatic switches that could quickly turn some of these characteristics on or off might allow the GA to construct simpler strategies to build on. Experimentation supported our intuition in this case (see Section 7.6.1 for discussion and supporting evidence for this side-point).

For the real-valued genes, we used Gaussian mutation, with a standard deviation of 10% of the parameter's allowed range, and for the boolean genes, we used simple bit-flip mutation. Crossover was performed only at the per-gene level – for simplicity, we treated the binary genes and real genes the same during crossover, and we did not employ more sophisticated real-valued crossover mechanisms (such as those proposed by [Ballester & Carter, 2004b]).

Settings	*corr*	L^0	L^1	L^2	L^∞
original	0.993 (0.0012)	22.8 (0.4)	1392.2 (123.8)	527.5 (47.6)	345.1 (35.0)
GA-*corr*	**0.999** (0.0002)	22.5 (0.6)	**409.4** (74.8)	**113.8** (24.7)	**61.9** (18.4)
GA-L^0	0.619 (0.0021)	23.0 (0.0)	7370.9 (56.9)	3074.5 (13.7)	2579.5 (15.4)
GA-L^1	0.996 (0.0011)	22.9 (0.3)	991.7 (158.0)	282.9 (37.7)	180.8 (19.2)
GA-L^2	0.995 (0.0007)	22.6 (0.5)	1150.5 (87.8)	323.6 (21.9)	156.8 (13.0)
GA-L^∞	0.991 (0.0010)	23.0 (0.0)	1694.7 (89.8)	436.9 (22.4)	172.2 (12.5)
Each cell gives the mean (and stdev) from 30 *replicate simulations.*					

Table 7.1. Calibration measure cross-comparison for the *toy problem*. The best GA-found parameter settings when optimizing using each calibration measure were evaluated against the target data using all five calibration measures. GA solutions were also compared to the *original* settings that were used to generate the target data. The best calibration values for each column are shown in bold (correlation is maximized, whereas the L^p error measures are minimized). (There was no clear best L^0 measure.)

For the fitness function, we used one of the calibration measures listed above in section 7.4.1, comparing simulated results from our ABM against the empirical traffic distributions for the same network.

In general, we ran 30 repeated searches using the GA for each calibration function and for each dataset (the toy problem and real world problem) . We ran the search for 200K fitness evaluations (model simulations) for the toy scenario, and 100K fitness evaluations for the comScore January dataset (the much larger dataset and longer simulation run-time necessitated running shorter searches). After the evolution finished, we chose the best search result from each of the 30. This gave us 5 parameter sets, P, for each problem; one for each error measure.

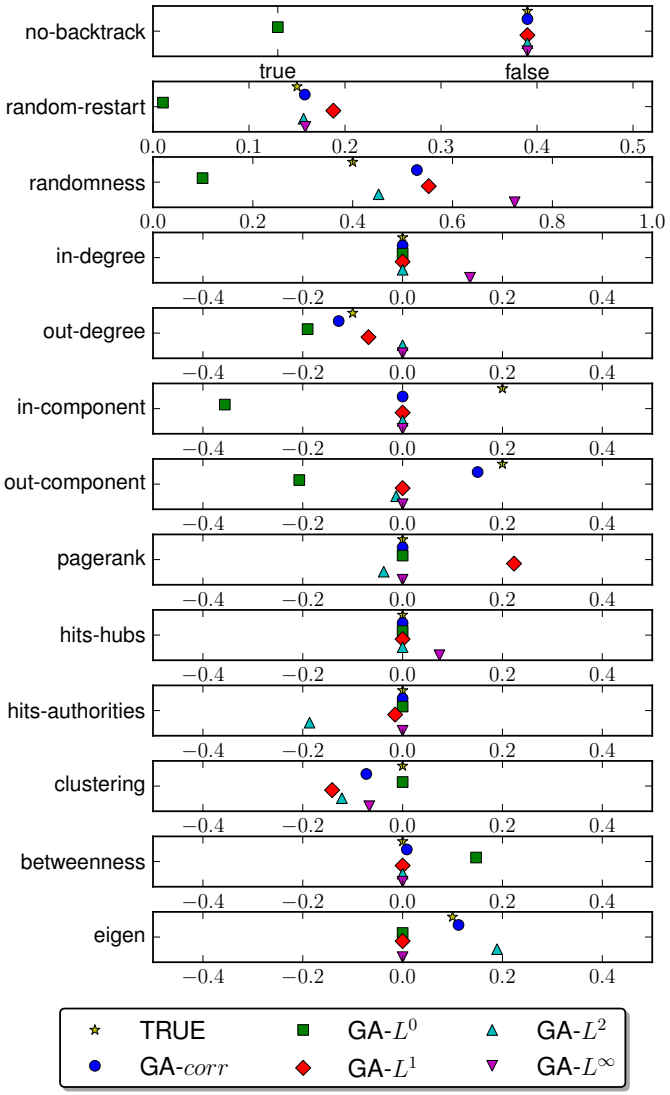


Figure 7.3. Parameter settings for the best individuals from the best GA-searches for each of the five calibration measures, for the *toy problem*.

7.5. Results and Discussion

7.5.1. Toy problem

The best results for the toy problem are given in Table 7.1, and the parameter settings that achieved those results are shown in Figure 7.3.

Finding 1: We were surprised to discover that the GA was able to find parameter settings that matched the target data *better* than the original parameter settings that were used to artificially generate the target traffic data. For instance, the average correlation measure from 30 runs with the best settings found in the GA-*corr* searches was slightly better than the correlation when the model was run 30 times with the *original* settings. Similarly, better L^1, L^2, and L^∞ error measures were achievable with the GA's settings than with the *original* settings. How is this even possible? The finding relies on the fact that the model is stochastic - different random choices by the surfer agent result in some variation in traffic distributions among the links. If the surfer agent was allowed to run for an infinite number of steps, the traffic distribution would converge to a steady state. However, after $10,000$ link-follows, results can still vary, meaning that when running the model with the original settings, and trying to match the data generated from one specific run, you do not automatically get a perfect matching.

Finding 2: The L^0 error measure performed very poorly (mean L^0 error of 23). In fact, L^0 error measurements are out of a maximum of 23, which corresponds to failing to perfectly match the traffic of any of the 23 links in the graph, and regardless of the error measure this value always neared 23. Even when the GA was searching for the best L^0 error-value, the parameter settings it found failed to match the data on any link on any of the 30 runs. On the other hand, some of the best calibrated parameters using other methods (e.g., GA-*corr*) managed to match exact traffic values occasionally. The poor performance on the GA-L^0 search can be attributed to its providing an insufficient search gradient, as well as the objective being very hard to achieve (as evidenced by the generally poor L^0 values obtained by all searches). Compared to the other L^p measures, optimizing the L^0 measure

is in some sense closer to a needle-in-a-haystack function and it should probably be avoided in GA-based calibration for this reason.

Finding 3: In this experiment, the GA-*corr* search proved to be the clear winner. As shown in Figure 7.3, the GA-*corr* parameter settings were similar, but not identical to the original (TRUE) settings that generated the toy dataset traffic distribution. Quite surprisingly, searching for good correlation yielded parameter settings that also provided lower L^1, L^2, and L^∞ error measures than the parameters discovered when attempting to optimize for those quantities directly. In other words, the *correlation* calibration measure served as the most effective fitness function for this problem, regardless of which calibration measure you were most interested in. Thus, the GA-*corr* parameter settings Pareto dominate the other parameter settings – that is, these parameter settings are as good or better than the parameter settings found using other fitness functions, on *every* calibration measures. This suggests that the correlation fitness function somehow smooths out the fitness landscape and more directly leads the population towards a more fruitful area of the search space than the L^p-based error measures do. In any case, this result was quite surprising to us, since we expected that GA-searches for a specific calibration measure (with the exception of the L^0 measure, which we anticipated might fail) would excel in optimizing its own value (though possibly at the expense of other calibration measures). This would be an extremely important and useful discovery if the superiority of correlation-based calibration were generally true; however, this is not always the case (as will be demonstrated by the experiments in the following section).

7.5.2. Real World Problem

7.5.2.1. Training set. We now turn our attention to the problem of matching empirical web traffic on our substantially larger dataset, where it is unknown how good of a calibration we can expect. The results from both the training and test set error measure comparisons are presented in Table 7.2, and the values of the best parameters found in each search are given in Figure 7.4.

Finding 1: Unlike in the toy problem, the GA is able to make some progress optimizing the L^0 calibration measure, as evidenced by it finding parameters yielding lower mean L^0 than the searches optimizing correlation or the other L^p norms. This makes some sense, because whereas the toy problem only had 23 links that could either match or not match, the comScore-January network had 3113 links, allowing the L^0-based fitness function a little more possibility to provide a search gradient. However, an L^0 measure of 2811 means that the simulated traffic did not match exactly on 2811 links, out of the 3113 links in the network. In other words, even in the best case, the GA was only able to find settings that could match about 10% of the network's links on average, while leaving 90% mismatched. This underscores the fact that requiring perfect matching of historical data is a harsh criteria for calibration, at least in our given scenario. In some cases, perfect matching of real-world data may be feasible, but even in such cases, we predict that calibrating using either correlation or an L^p norm where $p > 0$ will provide more information to the GA and permit more efficient search.

Finding 2: Unlike the toy problem where all the model results, regardless of the error measure, match well with the data using the correlation measure, in the real world dataset those parameter settings which were specified using one of the other error measures do not

generate model results which match well using the correlation measure. There are in fact statistically significant different results on all of the measures, but nowhere is this difference more striking than the correlation measure. The model data generated using the correlation error measure, correlated twice as well with the data as any other model data set that was generated. In the case of L^2, in particular, the data was almost uncorrelated with the real world data, indicating that a random result would have done almost as well as the results generated by the L^2 measure. This seems to indicate that correlation is definitely capturing a very different element of the matching problem than any of the other measures.

Finding 3: Unlike the toy problem where *correlation* was Pareto dominant, in the real world problem no error measures dominated all other error measures (though L^2 dominated L^1). Given that there is no pure dominant measure, researchers must be careful as to which measure to choose when calibrating their models. This is especially true given that Figure 7.4 shows that the actual parameter settings discovered by the various measures were not too different, and yet those different settings had a large impact on the error measure scores. As we discussed in Section 7.4 these different measures all take into account different choices, researchers should use L^0 if they are interested in the quantity of errors, and L^∞ if they are interested in magnitude of the largest error, but clearly there is no measure that will subsume all the others. Moreover, researchers should be aware of these measures and what effect each one has on the calibration effort when adjudicating the results. For instance, a paper which focuses primarily on L^∞ could very well be covering up that though the largest error was small, every single matching element was incorrect, and vice versa for L^0.

7.5.2.2. Testing set. Finally, we take the parameter settings, P^*, calibrated on the January data, R_{train} and examine the results of the error measures on a dataset collected 11 months later in December, R_{test} using the same set of five error measures.

Finding 1: As before, the L^0 measures are generally poor, indicating an inability to calibrate well with the data using the criterion that the simulated traffic value must match the empirical value exactly. This again begs the question of whether this measure is too difficult for complex problems. In fact there may by necessity be a trade-off where any model that is able to fit this measure well, will also not be very generalizable. The best model for this measure may very well be a model which does nothing but specify the exact values of all of the links, which is a model that could not be applied to any other network or dataset.

Finding 2: The L^p results are not substantially worse on the December data than they are on the January data. For instance, for the L^1 measure the error only goes up by 10%, though the increase in error appears to go up as you move toward L^∞, which indicates that the model is matching the same quantity of data points, but the magnitude of the worst difference is growing. Given the fact that the model was trained on a different dataset (some empirical results indicate that these two datasets have very different traffic patterns), this indicates that the parameter settings that the GA discovered using the various error measures on the January dataset are somewhat generalizable.

Finding 3: The relationship between the results within the correlation measure has changed. Though L^1, L^2, and L^∞ all do worse with regards to the correlation measure than they do on the January dataset, L^0 actually does better in terms of matching correlation values than it does on January dataset and approaches the correlation value achieved when the fitness function is in fact the correlation measure. This seems to indicate that though the L^0 measure is a difficult error measure to use as calibration the parameter settings that it generates may be useful with regards to other error measures.

Settings	*corr*	L^0	L^1	L^2	L^∞
GA-*corr*	**0.72** (0.0002)	2912 (15)	44364 (73)	7246 (49)	6190 (45)
GA-L^0	0.33 (0.0068)	**2811** (17)	42364 (135)	4198 (40)	2906 (49)
GA-L^1	0.30 (0.0048)	2836 (13)	42744 (67)	5977 (39)	3406 (51)
GA-L^2	0.02 (0.0039)	2883 (12)	**40499** (104)	**3197** (4)	2211 (2)
GA-L^∞	0.20 (0.0056)	2959 (10)	47293 (84)	3527 (17)	**1743** (23)
Calibration results on the comScore December testing data.					

Settings	*corr*	L^0	L^1	L^2	L^∞
GA-*corr*	**0.41** (0.0014)	2843 (14)	48843 (64)	9515 (46)	5260 (48)
GA-L^0	0.37 (0.0024)	2799 (15)	45482 (114)	6792 (48)	4584 (67)
GA-L^1	0.24 (0.0031)	2806 (13)	46145 (93)	7513 (40)	3915 (52)
GA-L^2	0.03 (0.0058)	2860 (14)	**44480** (109)	**4774** (5)	**3324** (5)
GA-L^∞	0.08 (0.0029)	2923 (12)	50911 (75)	5238 (17)	3343 (0)
Calibration results on the comScore December testing data.					

Table 7.2. Calibration measure comparison on comScore training and testing datasets. Each cell gives the mean (and stdev) from 30 replicate simulations. The best GA-found parameter settings when optimizing using each calibration measure on the January training data were evaluated against the January data (top) and the December data (bottom) using all five calibration measures. The best calibration value for each column is shown in bold (correlation is maximized, whereas the L^p error measures are minimized). (For December, there was no clear best L^0 measure.)

7.6. Genetic Algorithm Search Dynamics

7.6.1. Impact of epistatic chromosomal interactions

As mentioned in Section 7.4.2 above, we purposefully introduced epistatic interactions into the GA's genotype with the goal of improving performance when searching through the high dimensional parameter space. This also had the effect of causing the GA to more frequently sample simpler ranking strategies, which is beneficial from the perspective of Occam's razor. We placed each boolean switch gene adjacent to the gene for the coefficient that it epistatically controlled in order to promote linkage and make it more likely that the switch and its coefficient would be inherited together during crossover.

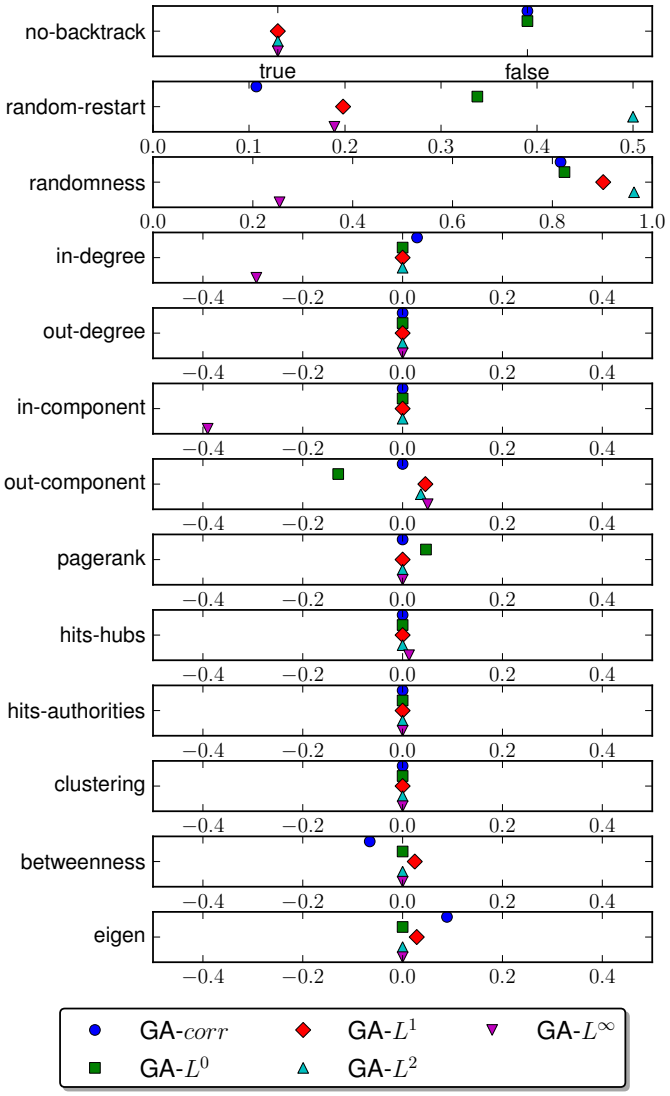


Figure 7.4. Parameter settings for the best individuals from the best GA-searches for each of the five calibration measures for the January dataset.

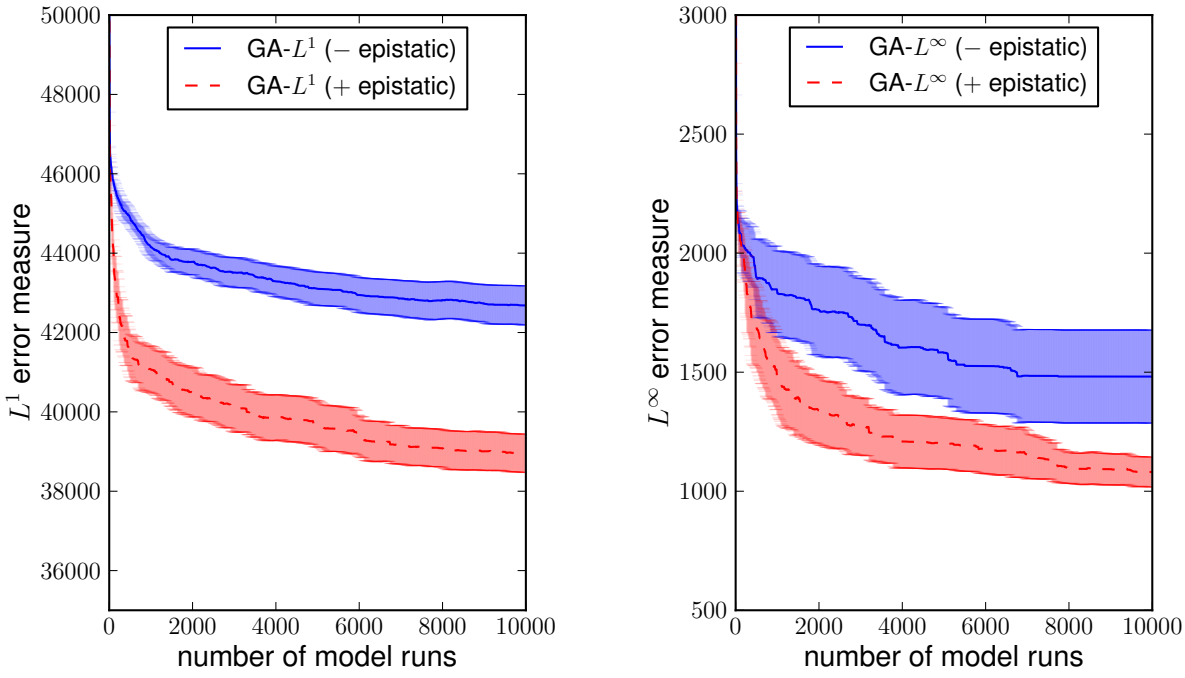
When working with genetic algorithms, one quickly learns not to trust one's intuitions; the road to search performance purgatory is paved with seemingly good intuitions. So despite the arguments made above, it was not clear that the engineered epistatic interactions would be beneficial. In fact, one could make a counter argument that they might be harmful

because the addition of the switches increased the size of the genome, and thus the size of the search space, which could plausibly contribute to worse performance.

To empirically settle the matter, we performed 30 genetic algorithm searches both with and without these epistatic switches, for the L^1, L^2 L^∞, and *corr* calibration measures, and recorded the average search performance in each case. In no case was the performance significantly better without the epistatic switches, and in several cases the switches resulted in a large and significant improvement, as shown in Figure 7.5. These results confirmed that the addition of boolean switches for epistatic interactions in the genotype can significantly increase the GA's performance. It could be that for this problem better solutions tend to involve fewer ranking coefficients, and thus the introduction of the switches biased the search toward more fruitful areas. Alternatively, the ability of the GA to manipulate the complexity of the problem on the fly (by turning switches on or off) may have contributed to the GA's success. In either case, this relatively simple addition to the genotype to promote epistatic interactions had a noticeable performance benefit, and it may provide a generally useful technique for similar ABM exploration situations.

7.6.2. *Deception* in real-world fitness functions

Significant research in genetic algorithms has focused on the characterization of those types of problems that are difficult for genetic algorithm (or other metaheuristic search techniques, like hill climbers) to solve. In particular, this led to the notion of "deceptive fitness functions" [D. Goldberg, 1987; Whitley, 1991; Horn & Goldberg, 1994], which are (broadly speaking) fitness landscapes that have local optima with large basins of attraction that tend to lead the search process away from superior global optima. While there has been some criticism of specific formulations of deception and its analysis with relation to genetic algorithms



(a) L^1 (b) L^∞

Figure 7.5. For several of the calibration measures (such as the L^1 and L^∞ searches shown here), the GA search performance was significantly improved by the use of epistatic switches controlling whether certain model parameters were allowed to vary or not.

performance [Grefenstette, 1992a; Forrest & Mitchell, 1993], it remains an important concept in the study of metaheuristic search and optimization. Most studies of *deception* have focused on artificially constructed functions (e.g., N-bit traps,), and there have been informal claims that deception might be irrelevant to real-world problems [Jones & Forrest, 1995]. Thus, it seemed noteworthy when we discovered evidence that was highly suggestive of a deceptive fitness function, when performing a follow-up experiment.

Specifically, we ran additional genetic algorithm searches directly on the testing dataset (December comScore) to determine the best calibration that was achievable on that dataset.

In general, these experiments confirmed that there was better calibration possible than had been achieved using the training set calibrated parameters. This was an expected result, and not particularly interesting. However, in the process of examining those results, we noticed an interesting pattern among the best solutions found when calibrating for *correlation* on this December dataset. As shown in Figure 7.6, the results generally fell into two clusters: those that achieved decent correlation (around 0.7) and those that achieved better correlation (around 0.8). Furthermore, these clusters in fitness value also corresponded to clusters within the parameter space. Several parameters varied between the clusters, but an easily distinguishable feature was that the better fitness cluster had no-backtrack $= FALSE$, whereas the worse fitness cluster had no-backtrack $= TRUE$. In essence, these groupings represent qualitatively different parameter settings, not merely quantitative variation around similar parameter settings. The other interesting facet was that the vast majority of searches ended at the inferior parameter settings. This shows that the representation of the search space and fitness function tend to lead the genetic algorithm to a suboptimal solution (perhaps because fitness improvement is easier early on in the search process when no-backtrack $= TRUE$).

These results indicate that for this specific problem, we are dealing with at least a mildly deceptive fitness function. For that matter, if a fitness function were strongly deceptive enough, we might never even realize it, if the genetic algorithm never found the true optima, and was always led to the suboptimal solution. This is a fundamental issue, and there is no guaranteed way to avoid it. Various approaches have been suggested to help genetic algorithms cope with deceptive problems (e.g., messy GAs [D. Goldberg, Korb, Deb, et al., 1989]), and often changes in chromosomal representation for the genetic algorithm can sufficiently rearrange the fitness landscape to decrease deception. Moreover, while it is desirable

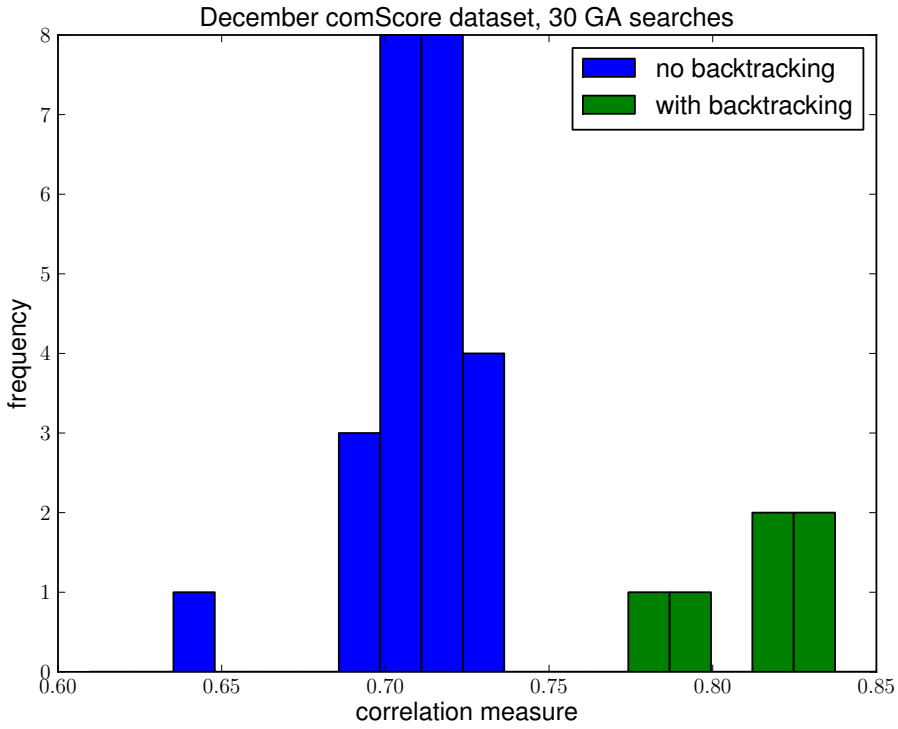


Figure 7.6. Distribution of fitnesses of the best parameter settings found when searching directly for best *correlation* with the December comScore dataset. Most of the 30 searches ended up at a suboptimal fitness peak, but a few were able to find a better solution (that included enabling backtracking for the web surfer agents). This strongly suggests that maximizing the correlation fitness function was a "deceptive" problem, in that local optima with large basins of attraction tend to lead the search process away from superior global optima.

to find the global optimum in a search space, there are many cases of ABM exploration where finding very good optima is sufficient to the task, regardless of whether they represent the absolute best parameter settings or not. In any case, it was interesting to witness evidence of a deceptive fitness function "in the wild" – that is, in the context of this real-world problem of model calibration, rather than in an artificially constructed functions where it is typically discussed.

7.7. Conclusion and Recommendations for Future Work

In this chapter, we explored the concepts of calibration and validation of an agent-based model using a variety of error measures in the context of an evolutionary algorithm. We have explored five different measures in the context of both a toy problem and a difficult real-world problem. We have shown that there is not an easily defensible Pareto optimal error measure that works in all cases, but we have illustrated what benefits and disadvantages each of these measures could have on model calibration using an evolutionary algorithm. To further explore trade-offs between different calibration measures, it might be useful to use multi-objective optimization to search for multiple calibration measures simultaneously, and thus reveal a Pareto front that will characterize trade-offs between different calibration measures [Narzisi et al., 2006]. As a side point, the benefit derived from expanding the genotype search space using epistatic boolean interactions also deserves further investigation. While genetic algorithms offer a promising technique for calibrating ABM parameters, one must give careful consideration to the choice of calibration measure. Different calibration measures provide varying levels of efficiency as fitness functions for performing this calibration, and can lead to varying results. Thus model analysts would be wise to try several different calibration measures when attempting to calibrate a model.

CHAPTER 8

Fitness Caching in Noisy/Stochastic Environments

"It's a good thing to have all the props pulled out from under us occasionally. It gives us some sense of what is rock under our feet, and what is sand."

– Madeleine L'Engle

"Nothing is built on stone; all is built on sand, but we must build as if the sand were stone."

– Jorge Luis Borges

Uncertainty is endemic to the human condition, as well as to the analysis of agent-based computer simulations. Consider the Wolf Sheep Predation model [Wilensky, 1997e] discussed in Chapter 3. Using the exact same parameter settings, one could run the model one million times, and every single time the wolf species could go extinct before 1000 model ticks have passed. Even so, we cannot be *absolutely certain* that on the next run the wolf species won't thrive indefinitely. This would be a very *unlikely* outcome – but we cannot be positive that it would not occur. This is the substrate on which we must build our model analysis – constructed on the "sands" of confidence and likelihoods, but never proved with the stable "rock" of certainty. Through effort and examination, uncertainty can be reduced, but it cannot be eliminated, and thus it must be faced and lived with. Likewise, intelligent search processes (such as genetic algorithms) must confront this uncertainty when they are applied to ABM exploration and analysis tasks. There are methods of dealing with uncertainty that

are more effective than others, and it is desirable to both understand and quantify how uncertainty impacts our attempts at model exploration and analysis. This is the subject of this chapter.

However, rather than focusing specifically on the domain of agent-based modeling, the work presented here is sufficiently general to apply to any situation where fitness evaluations are noisy but the uncertainty can be reduced by additional sampling. Compared to prior chapters, this chapter will place a stronger emphasis on a mathematical/analytic treatment of the problem, from first principles, although it will be accompanied by empirical results on traditional test-bed fitness functions. In order to make the analysis tractable, we will focus primarily on the impact of noise on the simpler random-mutation hill-climbing (RMHC) algorithm (which is similar to a $1 + 1$ evolutionary strategy), rather than facing the full complexity of the genetic algorithm. However, disregarding the crossover operator (which dynamically deforms the fitness landscape according to the genetic makeup of the current population), the GA's mutation operator navigates the same fitness landscape as the RMHC (and thus the GA resembles a parallel population-based hill-climber with dynamic reallocation of hill climbing resources). In short, although the GA is a more sophisticated search algorithm, theoretical analysis of the fitness landscapes for RMHC provides a reasonable first-order approximation for analyzing GA performance with the same fitness functions. More pragmatically, first-principles theoretical analysis of GA's performance has (thus far) proven infeasible without making significant simplifying assumptions.

Here is the central dilemma: given a noisy metric on an arbitrary agent-based model, how many times should one run the model in order to reduce the noise such that an adaptive search process (such as a genetic algorithm) can make positive progress through the space and arrive at good solutions? Without fitness caching, it may be possible to run only a

single replicate (thus getting a very noisy signal), but give the genetic algorithm a large population, so that even though many individual fitness comparison may be wrong, on average, the genetic algorithm may still make progress. Another way of considering this, is that an individual with poor fitness may last a while in the population by getting lucky with the noisy fitness evaluation, but over time it will probably be weeded out. However, when using fitness caching, the same point will never be evaluated again in future generations, since the cached value will then be used instead. With caching, rather than taking noisy measurements each time from the true fitness landscape, the situation is equivalent to having a frozen (incorrect) landscape, due to displacement of each point by noise. In this case, the choice of sampling replications is even more crucial, since it will determine not just how long it might take to reach an optimal value, but whether it is possible to reach it (and recognize it) at all. One extreme approach would be to run numerous replicates of the model and reduce noise to a very low level. However, this is wasteful, since evolutionary search does not require perfect values to operate efficiently. This chapter formally investigates this trade-off in an attempt to develop heuristics for quantifying search degradation due to noise, and help predict an appropriate level of sampling.

For many large-scale combinatorial search/optimization problems, meta-heuristic algorithms face noisy objective functions, coupled with computationally expensive evaluation times. In this chapter, we consider the interaction between the technique of "fitness caching" and the straightforward noise reduction approach of "fitness averaging" by repeated sampling. While both of these techniques are being used in practice, the interaction between them has not been previously investigated, and it is important to applications such as ABM exploration (among others). Fitness caching changes how noise affects a fitness landscapes, as noisy values become frozen in the cache. Assuming the use of fitness caching, we seek

to develop heuristic methods for predicting the optimal number of sampling replications for fitness averaging. We derive two analytic measures for quantifying the effects of noise on a cached fitness landscape (probabilities of creating "false switches" and "false optima"). We empirically confirm that these measures correlate well with observed probabilities on a set of four well-known test-bed functions (sphere, Rosenbrock, Rastrigin, Schwefel). We also present results from a preliminary experimental study on these landscapes, investigating four possible heuristic approaches for predicting the optimal sampling, using a random-mutation hill-climber with fitness caching.

8.1. Motivation

There are a number of problem features that universally pose challenges for all meta-heuristic search/optimization processes: predominant among these are noise/uncertainty, and the slowness of fitness evaluation (i.e., the time necessary to evaluate the objective function for any point in the search space). The presence of noise in a fitness function impedes making accurate comparisons between candidate solutions, or knowing how close the search process is to reaching a certain performance objective. In many cases, it is possible to use an average of many independent fitness function evaluations in order to reduce the noise. The length of time required for a single fitness evaluation can be significant, as it expands the length of the search by a direct multiplicative factor, and limits the number of evaluations possible for the search. Sometimes it is possible to use a less accurate surrogate fitness function, which can be evaluated more quickly, but at the cost of additional noise in the fitness estimates (for a survey of fitness approximation, refer to [Jin, 2005]). In general, it is impossible to eliminate both of these problem features, although there are many problems where trade-offs can be made between the two.

When fitness evaluation is particularly computationally expensive (e.g., in large complex simulations), it is sometimes attractive to cache fitness values for re-use, to save the cost of re-evaluating them again later. At least in some non-noisy optimization problems, this has been shown to be an effective approach for reducing total computational cost [Kratica, 1999; Kratica et al., 2001], and we believe there is potential for applying it to noisy search spaces as well. In this work, we apply a combination of formal and empirical methods to try to investigate the relationship between fitness caching and the noise reduction technique *fitness averaging by repeated sampling*. In noisy environments, too little sampling can make the search untenable, whereas too much sampling can be unacceptably slow. Somewhere in between, there exists an ideal number of sampling repetitions, or "sweet spot", where the search most efficiently reaches a desired fitness level. Assuming the use of fitness caching, and using only information that can be extracted from the fitness landscape with reasonable efficiency, we would like to be able to predict where this "sweet spot" will fall.

The basic intuition motivating this research is that some landscapes are much more sensitive to the effects of noise than others, with regard to movement through these landscapes. For instance, a landscape that contains large steep mountains may be easily traversed to values of high fitness, despite the presence of significant noise, whereas even a small amount of noise may cause a landscape comprised of gentle slopes to become unnavigable. It would be very useful to have an efficient method of assessing the robustness of a landscape with respect to noise, in order to choose an appropriate sampling rate when applying a meta-heuristic search technique to the problem. The current study investigates the correlation between the distribution of fitness gradients throughout the landscape and the deleterious effects of varying levels of noise on landscape traversal.

This chapter begins by situating the present work in the context of related research in the field. We then propose two measures to quantify the impact of noise on search processes within fitness landscapes: the probability that noise generates false local optima in the landscape, and the probability that noise will result in an incorrect choice when comparing two neighboring locations in the space. We offer mathematical expressions for these two measures, which are numerically confirmed by Monte Carlo simulations of the two respective probabilities, on a set of four well-known test-bed functions (sphere, Rosenbrock, Rastrigin, Schwefel). Next, we discuss how these measures could be used in heuristics for choosing an optimal sampling number for noise reduction. We then present the results of an experimental study where we empirically determine optimal sampling rates on the four test landscapes, given a straightforward local search technique (stochastic hill climber) that uses fitness caching, and compare the potential of four heuristic approaches to predict the "sweet spot" for noise reduction. The chapter concludes by presenting several avenues for possible future work in this vein.

8.2. Related Work

The beneficial effects of fitness caching (specifically for genetic algorithms) have been discussed by Kratica [Kratica, 1999], and also applied to a practical problem (plant location) in [Kratica et al., 2001]. In [Kratica et al., 2001], the authors note that one of the conditions for successfully applying fitness caching is a large evaluation time for the fitness function. One real-world example where fitness caching may be beneficial is the the optimization of simulation parameters, since complex simulations may require long running times. However, another aspect of many real-world optimization problems is the presence of

noise or uncertainty. For example, a recent instance of fitness caching [Stonedahl & Wilensky, 2011] (see also Chapter 4) used two meta-heuristic search algorithms (hill-climbing and genetic algorithms) to explore the parameter-space of several agent-based simulations of biologically-inspired flock formation. In this case, the multi-agent simulations were stochastic, resulting in noisy fitness evaluation; however, the interaction of fitness caching with the noise was not explored. The situation is similar for many related problems involving the exploration of multi-agent based simulation, including parameter optimization [Stonedahl et al., 2010] and calibration and sensitivity analysis [Stonedahl & Wilensky, 2010b] (see also Chapters 5 and 6). Moreover, while there is a potential benefit for fitness caching, even in noisy environments, we are unaware of prior work discussing the use of fitness caching in noisy/uncertain optimization problems, or examining the potential repercussions for search performance in detail.

Considerable research has been done in the general area of meta-heuristic search and optimization in noisy fitness landscapes, and it remains a topic of considerable interest. For example, recent work spans from developing efficient techniques of determining the best individual from a noisy population [Jaskowski & Kotlowski, 2008], to defining standard sets of noisy functions for benchmarking different optimization techniques [Hansen, Finck, Ros, & Auger, 2009b]. The volume and breadth of work in this area is beyond the scope of this thesis; for a comprehensive survey of noise/uncertainty in evolutionary algorithms, see [Jin & Branke, 2005].

It is worth highlighting some of the more closely-related research. One strand of research concerns the analysis of search spaces or fitness landscapes, such as the study of Kauffman's NK-landscapes [Kauffman, 1993; Kauffman & Levin, 1987], similarly inspired tunable landscapes [R. Smith & Smith, 2001], as well as search performance on such landscapes (e.g.,

[Merz & Freisleben, 1998]). Also particularly relevant is the work on adaptive walks through noisy fitness landscapes [Levitan & Kauffman, 1995]. Our work also pertains to adaptive walks (or local neighborhood-based search algorithms in general) in noisy landscapes, but with fitness caching the noise becomes frozen, as we will discuss later. Also, because our application interests are focused more on simulation parameter search rather than understanding of biological evolutionary processes, we chose to investigate landscapes based on real-valued optimization benchmarks (see Section 8.2 below). So-called "fitness evolvability portraits" [T. Smith, Husbands, Layzell, & O'Shea, 2002] appear to be another promising direction for fitness landscape analysis. While Smith et al. [2002] did not address issues of noise, in the future several of the ideas about characterizing the landscape at different fitness levels might be productively incorporated on the sampling with fitness caching problem we are addressing here.

Several prior works ([Fitzpatrick & Grefenstette, 1988], and more recently [Balaji, Srinivasan, & Tham, 2007]) have discussed/debated the relative merits of repeated sampling for noise reduction versus alternative methods, such as increasing population size. However, when fitness caching is used, separate individuals in a population-based search do not contribute independent fitness trials, so increasing the population offers no advantages in reducing the impact of noise. Rana et al. [1996] examine the effects of noise on search landscapes, in particular discussing the creation of *false local optima* and the soft annealing of peaks (or "melting" effect, as referred to by Levitan and Kauffman [1995]). Our current work is also interested in the creation of false local optima by noise, but the use of fitness caching changes both the character and consequences of such local optima (as we discuss in Section 8.3.1).

Our work also follows that of Hughes [2000], which derived analytic expressions quantifying the probability of one individual having a higher true fitness than another, given noisy fitness evaluation, in the context of both single and multi-objective evolutionary algorithms. Though several of the derivations are mathematically related, the measures we derive attempt to characterize the fitness landscape as a whole, rather than a single comparison.

In summary, this investigation is the first to discuss and analyze the effect of fitness caching in noisy fitness landscapes, and to develop preliminary heuristics for helping choose the most effective number of sampling repetitions in this case.

8.3. Theoretical Analysis

We will begin from a theoretical perspective, offering a formal description of the problem, and deriving several mathematical measures that may be useful, before we move on to more experimental methods.

In this work, we will assume the presence of additive Gaussian (normally distributed) noise with mean 0. The situation we are concerned with is the repeated sampling of a noisy fitness function, and as a result of the Central Limit Theorem, the shape of the noise distribution will always approach a normal distribution when a reasonably large number of samples is used. However, the mathematical derivations we present below could equally be applied to other noise distributions, although the resulting expressions may be symbolically and/or computationally cumbersome. If the mean value of the noise is unknown (and nonzero), then regardless of any approach, it impossible to determine the true expected value of the fitness landscape at any point; thus we will only consider unbiased noise with zero mean. We will also assume that the variance of the additive noise is uniform across the search space – while this is not always the case, it serves as a reasonable first-order approximation to simplify

the analysis. The extension of considering noise with location-dependent variance is left as future work.

We will also make the simplifying assumptions that there is ample memory such that all encountered fitness values will be cached and are never cleared, and that the computation time required for the caching is negligible compared to the time required for fitness evaluation. These assumptions are realistic when fitness evaluation is particularly time-consuming, such as when optimizing complex simulations with lengthy run-times. In this case, high-capacity disk-based caching becomes a feasible approach, when the disk-access time for reading a cached fitness value may be orders of magnitudes smaller than the run-time of the simulation.

8.3.1. Derivation of Measures

Let us consider a "true" (noiseless) landscape function L which has been obscured by some amount of additive noise (N), which is drawn from a normal distribution with mean 0 and standard deviation of σ ($N \sim \mathcal{N}(0, \sigma^2)$).[1] We will assume the neighborhood-based search, where the task is minimization (find x s.t. $L(x)$ is a minimum). Without fitness caching, each time a search algorithm evaluates a point x_1 in the search space S ($x_1 \in S$), a new fitness value $L(x_1) + N$ is returned, where N is independently drawn from $\mathcal{N}(0, \sigma^2)$. Let x_2 be a neighbor of x_1, such that $L(x_2)$ is greater than $L(x_1)$ by a positive amount ϵ ($L(x_2) = L(x_1) + \epsilon$). This means that if the search process was repeatedly choosing whether to move between x_1 and x_2, it would (probabilistically) end up at x_1. With fitness caching, this is not the case. Once fitnesses for x_2 and x_1 have been chosen, they are fixed, or *frozen*. This caching is effectively the same as reading values from a new "frozen" noisy landscape

[1] In the context of real-world problems, it may be confusing to think of there being a "true" fitness landscape with noise being added to it; alternatively, L may be viewed as the true *expected value* of the noisy function.

L_n, which is generated from L by adding N ($N \sim \mathcal{N}(0, \sigma^2)$ to every location in X. If the fitness value $L_n(x_2)$ turns out to be smaller than $L_n(x_1)$, then noise has caused a comparison between two points to now be wrong (we will denote this as a "false switch"). This freezing effect means that when fitness caching makes the impact of noise more serious. Furthermore, rather than noise having a positive "melting" effect that can help a search process escape local optima (as further discussed in [Levitan & Kauffman, 1995; Rana et al., 1996], and as is implicit in the design of simulated annealing), fitness caching causes any new local optima that are created by the noise to be "frozen" in place. We will denote local optima that are present in L_n, but not present in the original L as "false optima".

When faced with a new landscape to be searched, we do not know what the landscape looks like. However, it is possible to probe the landscape for some information, before starting a search process. Let us assume that we can obtain a reasonable estimate of the true ϵ-distribution within the landscape. That is, we would like to capture the distribution of fitness differences between neighboring points $(L(x_i) - L(x_j) \forall (x_i, x_j) \in S^2$ s.t. x_i and x_j are neighbors in the space). We will denote the probability density function (pdf) for this ϵ-distribution as $P(\epsilon)$. Note that the $P(\epsilon)$ distribution is symmetric with respect to 0 because the neighbor relationship is symmetric. (Monte Carlo sampling from L_n will give an estimate of the noisy ϵ-distribution, which may be a tolerable approximation of the true ϵ-distribution, or may need to be corrected for noise.)

Given the pdf $P(\epsilon)$, we will now derive expressions for the likelihood of noise creating false switches and false optima, in terms of the standard deviation of the noise (σ).

For convenience, we will denote the pdf for the Gaussian distribution with mean value, μ, and standard deviation, σ by $f(x, \mu, \sigma)$, defined as follows:

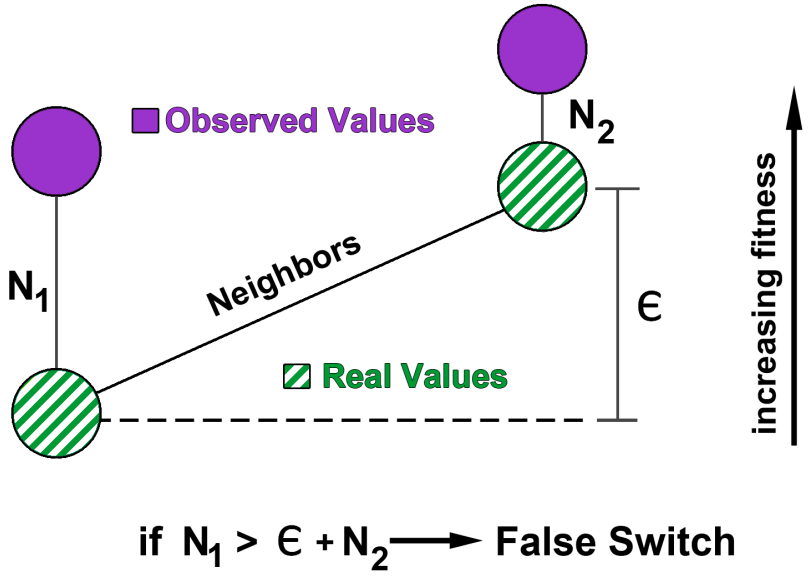


Figure 8.1. This figure illustrates variables used to determine the existence of a false switch. N_1 and N_2 represent the added noise to the original nodes, and ϵ represents the vertical distance between the two original neighbors. False switches occur whenever N_1 is greater than $\epsilon + N_2$.

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)} \tag{8.1}$$

8.3.1.1. False Switch Probability. In Equation 8.2 the inner integral represents the probability of the noise added to L(x_2), N_2 being less than the noise added to L(x_1), N_1. The inner two integrals (together) represent the probability of a false switch for a given difference between neighbors' real fitness values, ϵ. The outermost integral (integrating across all possible ϵs) computes the probability of a false switch for a given ϵ-distribution P(ϵ).

$$2\int_0^\infty P(\epsilon)\left(\int_{-\infty}^{\infty} f(N_1, 0, \sigma)\left(\int_{-\infty}^{N_1} f(N_2, \epsilon, \sigma) dN_2\right) dN_1\right) d\epsilon \tag{8.2}$$

Equation 8.2 can be simplified to Equation 8.3, where Erf denotes the Gaussian error function. It has been remarked in certain contexts [Hughes, 2000] that the Gaussian error function (Erf) is computationally very time-consuming to compute, and that more efficient (though slightly less accurate) approximations may be desirable. However, our approach is to derive a measure that will characterize the robustness of the fitness landscape as a whole. This is essentially an *offline* calculation which will be completed once before initiating a search process, rather than an *online* calculation that must be run repeatedly during the search process. Furthermore, since fitness caching is being used, there is an implicit assumption that evaluating a single point from the fitness landscape takes orders of magnitude longer than other operations, and the efficiency of numerical approximations is not a primary concern.

$$\int_0^\infty P(\epsilon)\left(1 - Erf\left[\frac{\epsilon}{2\sigma}\right]\right) d\epsilon \tag{8.3}$$

8.3.1.2. False Optima Probability. In order to obtain an analytic formula for the probability of creating false optima, we must make the additional simplifying assumption that the distribution $P(\epsilon)$ is the same throughout the space – i.e., at every x, $P(\epsilon)$ is the same regardless of $L(x)$.

For an arbitrary noise distribution ($P(N)$), the probability of being a local optimum in L_n is given by Equation 8.4.

$$\int_{-\infty}^{\infty} P(N_1)\left(\int_{-\infty}^{\infty} P(\epsilon)\left[\int_{-\infty}^{-\epsilon+N_1} P(N_2)dN_2\right] d\epsilon\right)^n dN_1 \tag{8.4}$$

Similarly, the probability of a given point being a local optimum in both L and L_n is given by Equation 8.5.

$$\int_{-\infty}^{\infty} P(N_1) \left(\int_{-\infty}^{0} P(\epsilon) \left[\int_{-\infty}^{-\epsilon+N_1} P(N_2) dN_2 \right] d\epsilon \right)^n dN_1 \tag{8.5}$$

False optima are points that appear as local optima after noise is applied, but were not local optima before noise, thus the probability of being a false optimum is calculated by subtracting Equation 8.5 from Equation 8.4. Equations 8.4 and 8.5 were for arbitrary noise distributions, but since we are assuming all noise is additive Gaussian noise, we can transform them into Equations 8.6 and 8.7 respectively.

$$\frac{1}{2} \int_{-\infty}^{\infty} f(N_1, 0, \sigma) \left(\int_{-\infty}^{\infty} P(\epsilon) \left[1 + Erf \left[\frac{-\epsilon + N_1}{\sigma\sqrt{2}} \right] \right] d\epsilon \right)^n dN_1 \tag{8.6}$$

$$\frac{1}{2} \int_{-\infty}^{\infty} f(N_1, 0, \sigma) \left(\int_{-\infty}^{0} P(\epsilon) \left[1 + Erf \left[\frac{-\epsilon + N_1}{\sigma\sqrt{2}} \right] \right] d\epsilon \right)^n dN_1 \tag{8.7}$$

Given $P(\epsilon)$ (the probability density function for the ϵ-distribution of a fitness landscape), we now have closed-form expressions for the probabilities of a "false switch" occurring between any two neighboring points, and the probability of any given point becoming a "false optimum."[2]

[2]Despite being closed-form mathematical expressions, numerical integration approaches will generally be required, especially since $P(\epsilon)$ may be any arbitrary pdf.

8.3.2. Fitness Landscapes

The abstract elegance of formally deriving mathematical measures or descriptive statistics about fitness landscapes must be grounded by the study of concrete fitness landscapes. This partially serves to validate the derivations, but more importantly it helps us judge the appropriateness of any simplifying assumptions that were made in order to make the mathematics tractable.

For our fitness landscapes, we selected four noiseless fitness functions that are often studied in the context of real-valued black-box optimization, and which exhibit differing landscape features (such as multi-modality/nonconvexity). Specifically, we chose the sphere, Rosenbrock, Schwefel, and Rastrigin functions (adapted from [Hansen, Finck, Ros, & Auger, 2009a]). These noiseless landscapes are assumed to be the "true" underlying functions, which we will combine with varying levels of additive Gaussian noise to create the "obscured" noisy fitness landscapes that must be searched. Surface plots of the 2-dimensional versions of these fitness landscapes are shown in Figure 8.2, shown for illustrative purposes to communicate the general shape of these spaces. All results presented here used the 10-dimensional version of these functions, where each dimension was discretized on the domain $[-5, 5]$ at a resolution of 0.05, creating a discrete search space of size $201^{10} \approx 1.1 \times 10^{23}$. The general mathematical function to generate the N-dimensional case for each landscape is displayed below the graphics in Figure 8.2.

In Figure 8.3, kernel density distribution plots[3] show the ϵ-distributions (distribution of differences between the "real" fitness values at neighboring locations in the fitness space)

[3]Kernel density distribution plots provide a way to visualize distributional information that avoids the artifacts caused by bin-size choices in histograms.

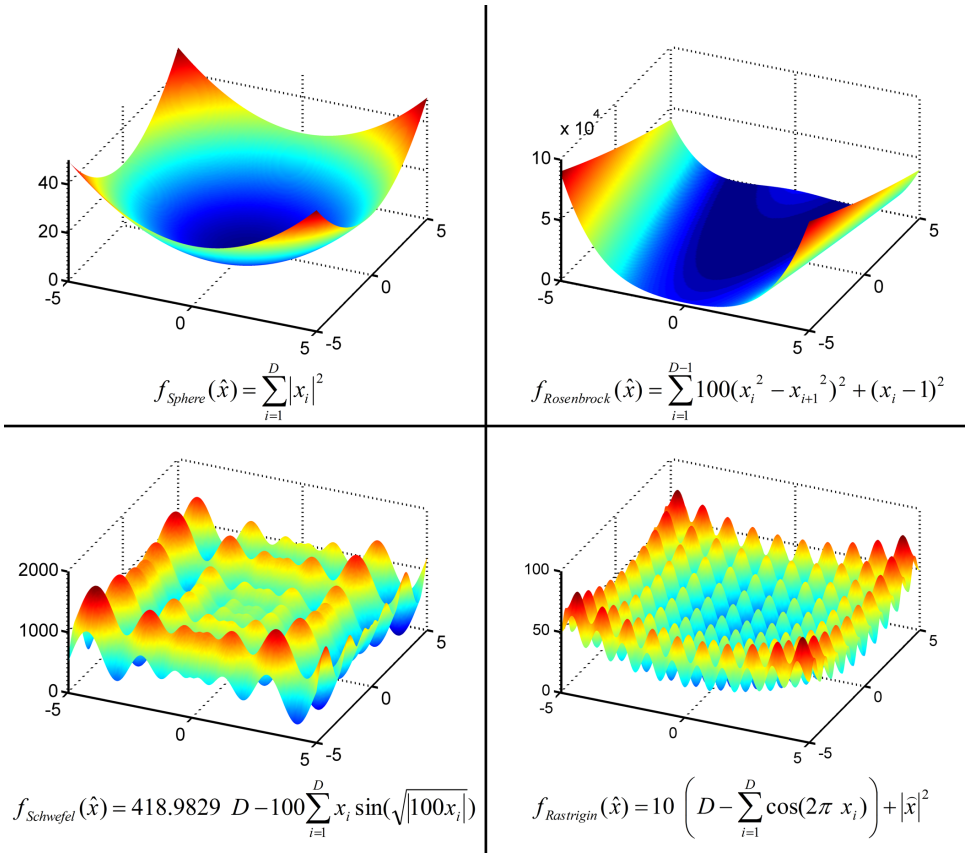


Figure 8.2. This figure shows 2-D versions of the sphere, Rosenbrock, Schwefel, and Rastrigin functions we used as our fitness landscapes. The equations are shown below each plot.

for each of these landscapes. Note that the different distributions vary significantly in shape and range of values.

8.3.3. Empirical Measure Validation

We predicted the number of false switches and false optima in each fitness landscape using the measures defined in Section 8.3 above and an approximate ϵ-distribution defined by sampling 5000 differences between neighbors' real fitness values. Then we observed the real probability of false switches being created by noise by testing 10,000 pairs of neighboring points, which

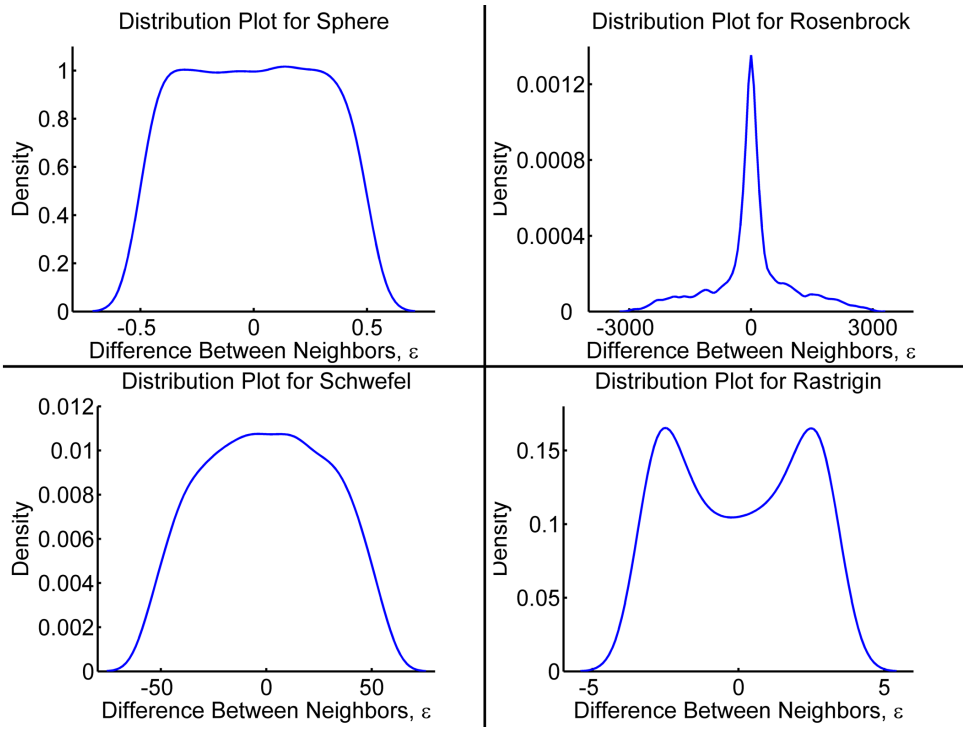


Figure 8.3. This figure shows the ϵ-distribution (fitness differences between neighboring locations) for each fitness landscape.

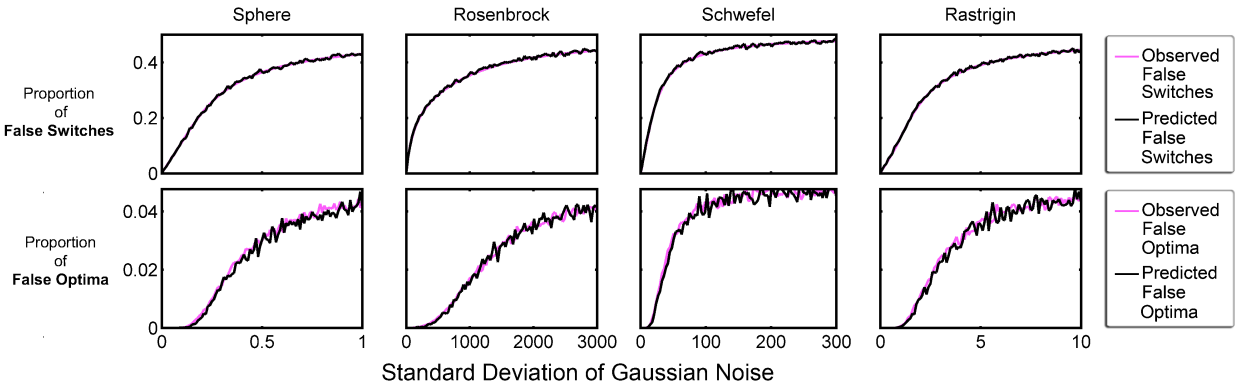


Figure 8.4. We predicted the probabilities of false switches and false optima occurring using the measures presented in Section 8.3 and observed the actual probabilities that each occurred by adding various amounts of noise to each function and evaluating the resulting proportions of false switches and false optima.

were evaluated before and after varying amounts of Gaussian noise was added. Similarly, we used a Monte Carlo method (testing 10,000 points) to estimate the real probability that a point becomes a false optimum as a result of differing magnitudes of Gaussian noise. As shown in Figure 8.4, the formulas we derived for these two measures closely approximate the directly observed measures.

8.4. Experiments

We are further interested in whether these or other simple measures can be useful in predicting the performance of an evolutionary search technique on a noisy landscape. In particular, it would be most useful to be able to choose the number of times a noisy function should be evaluated and averaged, to enable a search mechanism to reach very good locations in the space with as few function evaluations as possible. Specifically, we ran experiments at varying noise levels to determine the number of evaluations required by a stochastic hill climber to reach an average fitness value that is in the best 0.0001% of the landscape. These numbers of evaluations are then scaled by the number of times the function would need to be evaluated to reach their respective noise levels. The pseudocode for the simple random-mutation hill climbing algorithm is given in Table 8.1.

The noise level (standard deviation of noise) for which the search progresses most rapidly is denoted σ_{ideal} (which will vary for each landscape). See Figure 8.5 for an illustration of this process.

We considered four heuristic methods for using a landscape's ϵ-distribution to predict σ_{ideal} and compared the number of evaluations required by the hill climber at each method's predicted σ_{ideal} to those required at the true σ_{ideal}.

Given a (memoizing) noisy landscape function L_n, and a function $neighbor(x)$ which returns a new location by increasing or decreasing x along a single randomly chosen dimension:
1. Let $x_{best} = \varnothing$ 2. Choose x randomly from S (the search space) 3. If $x_{best} = \varnothing$ or $L_n(x) < L_n(x_{best})$: Set $x_{best} = x$ 4. If evaluation limit exceeded: Return x_{best}. 5. If x has been compared to all of its neighbors and is a local minimum: Go to Step 2. 6. Let $x' = neighbor(x)$ 7. If $L_n(x') < L_n(x)$: Set $x = x'$ 8. Go to Step 3

Table 8.1. Pseudocode for a random-mutation hill climber, which restarts when stalled.

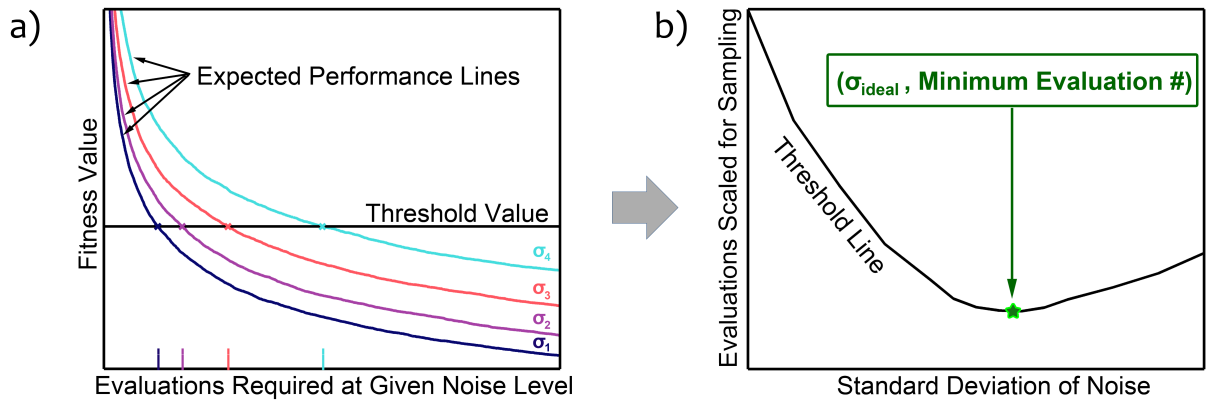


Figure 8.5. a) Each shaded line shows fitness values reached after some number of evaluations, for a given noise level, σ_x. Using this information we calculated the number of evaluations it took to reach a threshold value, and scaled this by the number of replicate evaluations required to reduce noise to the specified level (σ_x). b) This scaled number of evaluations is plotted at each noise level. We denote the noise level corresponding to the minimum number of evaluations as σ_{ideal}, which is the "sweet spot" target for noise reduction.

The four heuristics for predicting σ_{ideal} are listed below. In order to calibrate the heuristics, it was necessary to use scaling factors based on the true σ_{ideal} for the landscape. We

then tested the heuristics by applying them to each landscape in turn, in order to evaluate whether they could capture the differences between the landscapes.

- *Fixed Noise Level:* The geometric mean of the σ_{ideal} for each landscape is 1.91 and this constant noise value was used as the $\sigma_{Fixed\ Noise\ Level}$. This is the most naïve heuristic, as it treats all landscapes the same, without making use of the ϵ-distribution information at all. It is included mainly as a baseline for comparison.
- *Direct Ratio:* The geometric mean of the ratio of the median of each ϵ-distribution to the σ_{ideal} for each landscape is 3.97. We calculated $\sigma_{Direct\ Ratio}$ by dividing the median of each landscape's ϵ-distribution by this ratio.
- *False Switch:* The geometric mean of the proportion of false switch values corresponding to the σ_{ideal} for each landscape is 0.084. The standard deviation of noise which predicts a proportion of false switch value of 0.084 is the $\sigma_{False\ Switch}$
- *False Optima:* The geometric mean of the proportion of false optima values corresponding to σ_{ideal} for each landscape is 5.16×10^{-5}. The standard deviation which predicts this value is the $\sigma_{False\ Optima}$.

8.5. Results and Discussion

To compare these methods on each of the four landscapes, we calculate the *inefficiency ratio* as the number of evaluations required by each method's prediction for σ_{ideal} (i.e., $\sigma_{Fixed\ Noise\ Level}$, $\sigma_{Direct\ Ratio}$, $\sigma_{False\ Switch}$, $\sigma_{False\ Optima}$) divided by the number required at the true σ_{ideal}. Note that an inefficiency ratio of 1.0 would be a perfect prediction, and also that ratios higher than 20 have been cut off, due to computational constraints.

To summarize the performance results from Figure 8.6:

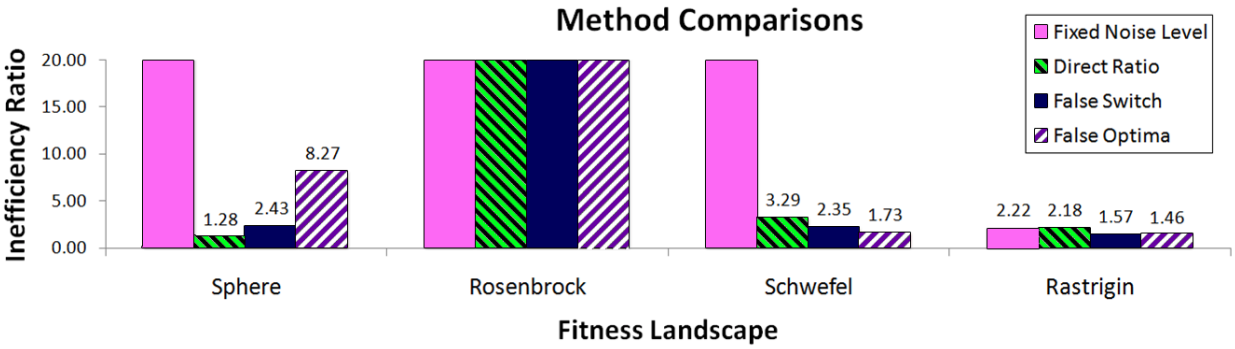


Figure 8.6. This figure shows how inefficient the standard deviation chosen by each method is by calculating the ratio of evaluations to that required at optimal noise level, σ_{ideal}. A perfect solution would have an inefficiency ratio of 1.0.

(1) None of the methods performed well on the Rosenbrock landscape. The Rosenbrock function is sometimes referred to as a "banana function" due to its long bending valley which must be followed to reach the global optimum. The failure to predict an optimal level of noise may be due in large part to the importance of traversing this valley, where the fitness gradient is not very strong. In other words, the initial sampling of the whole space to determine the ϵ-distribution is misleading, since a particular region of the space (the valley floor) is much more important for search performance than the space at large, and requires lower noise values to traverse.

(2) The *fixed noise level* method performed quite poorly on all but one landscape. In general, this is not too surprising. We expect that different landscapes will require different optimal noise levels, and choosing a fixed level value to apply to all landscapes is unlikely to perform well.

(3) There is no clear winner among the other three methods: the *false optima* and *direct ratio* methods were each best on certain landscapes, but the *false switch* method also generally performed well. This result is somewhat disappointing, in that heuristics

using our derived metrics (*false switches* and *false optima*) do not have a strong advantage over the simpler approach (*direct ratio*) of scaling by the median value from the ϵ-distribution.

While these results are not decisive, it is somewhat encouraging that the three methods using information from the ϵ-distribution serve as better predictors than the most naive approach. This shows that the heuristics used are at least partially correlated with choices for σ_{ideal}, and perhaps improved mappings may be developed along similar lines, in order to offer prescriptive guidelines for choice of sampling repetitions based on this information.

8.6. Future Work and Conclusions

The experimental results we have presented are based only on an examination of four fitness landscapes, which is too small to be a good representation of the types of fitness landscapes encountered in real problems. Furthermore, it has been argued that some of these particular test landscapes may not be the most appropriate choice for benchmark functions for evolutionary algorithms [Whitley, Mathias, Rana, & Dzubera, 1995]. Accordingly, further studies along similar lines are called for, involving a greater diversity of noisy fitness landscapes.

However, perhaps a more significant challenge for the current approach is that the search performance on these landscapes appears to be significantly different enough that none of the heuristics we investigated served as a good predictor for all four landscapes. In particular, the failure to predict a good noise level for the Rosenbrock landscape merits further investigation. It is possible that a fundamentally different approach will be needed. One hypothesis is that knowledge of the global ϵ-distribution for a landscape is insufficient to make a good prediction of what the optimal noise level would be, and thus additional knowledge is required. This may

be because intelligent search techniques find relatively good solution areas quickly, and thus spend very little time in the large poor-performance areas of the space, in which case a more biased approach for sampling ϵ-distributions might be fruitful (e.g., taking inspiration from [T. Smith et al., 2002]). For instance, one could imagine running a sequence of searches, bootstrapping the ϵ-distribution from the points that were encountered by the previous search on the noisy landscape, thus refining the estimates for optimal sampling choices in later searches.

In addition to their role in meta-heuristic search processes, fitness landscapes also play an important role in the study of many complex systems, and may provide a lens for viewing adaptive or evolving systems in new and enlightening ways (c.f. Kauffman's work on evolutionary landscapes [Kauffman, 1993]). It would be interesting to investigate whether there are interdisciplinary implications for studying frozen noisy landscapes, in relation to processes that occur in real biological systems.

An improved understanding of the extent to which noise can be present in a fitness landscape without seriously inhibiting successful search and adaptation in that space is a broad but desirable goal, which would significantly advance the field of search/optimization when dealing with uncertain problems. Our present research provides some progress toward this goal in the specific context of fitness caching, but the path is far from clear, and significant work remains to be done in this direction. The lack of prior literature on fitness caching with noise may suggest either that the combination has not been given serious consideration, or possibly that fitness caching is not an advisable approach when dealing with noisy search problems. While we believe that in many cases it would still prove beneficial, this is ultimately an empirical question, and one that we hope will be resolved by future work using fitness caching in noisy environments.

In conclusion, this chapter offered a brief foray into the study of the interactions between noisy landscape sampling and fitness caching. We presented and verified analytic formulas for two measures that could be useful for predicting the impact of noise on the performance of fitness-caching neighborhood based meta-heuristic search processes in discrete fitness landscapes. We also explored several heuristics for choosing an optimal sampling level under these conditions, and while none of these heuristics offer perfect solutions to this problem, they could provide reasonable initial choices when there is no *a priori* information about what sampling level to use for an unknown fitness landscape. Additionally, they provide a starting place for developing better heuristics for this problem. However, further research is required before we can offer prescriptive recommendations for noise level reduction methodology. Some of the necessary research is carried out in Chapter 9, where varying levels of samples are used for noise reduction in the fitness landscapes associated with real ABM exploration tasks, using a wider variety of search algorithms (hill climbing, genetic algorithms, simulated annealing, and random search), and comparing the results both with and without fitness caching.

CHAPTER 9

Comparative Benchmarking in ABM exploration

"There is nothing like looking, if you want to find something... You certainly usually find something, if you look, but it is not always quite the something you were after."

– J.R.R. Tolkien, *The Hobbit*

"All generalizations are false, including this one."

– Mark Twain

As Tolkien reminds us, the act of searching for something does not ensure that you will find it. Often you may find something else instead, which may be interesting in its own right, but isn't what you set out to find. When exploring the parameter space of agent-based models, this can be beneficial, since each interesting finding may shed light on some aspect of model behavior. Nevertheless, there are many situations where you have a specific goal in mind (e.g., finding vee flock formations, or calibrating a model to real-world data), and you really do want to find parameters that best accomplish that specific task, rather than parameters that do something else. Thus, it is important to measure the efficacy of search methods in performing ABM exploration tasks. The case studies presented in Chapters 4–7 already showed that genetic algorithms were useful and effective for ABM exploration tasks, by digging deep into relevant research-caliber modeling problems. In contrast, the focus of this chapter is on breadth, not depth. This chapter provides the first large-scale comparative study of the genetic algorithm's performance on a wide variety of

models and tasks, with various levels of sampling, both with and without fitness caching, and judged against other blackbox metaheuristic search algorithms including random search, hill climbing, and simulated annealing.

No set of experiments in this domain can be truly *comprehensive*, as countless other agent-based models could be explored, and there is also no shortage of metaheuristic search algorithms (such as particle swarm optimization [Kennedy et al., 1995], and harmony search[Geem, Kim, & Loganathan, 2001]) that remain untested. However, unlike prior studies that focused on a single model or a single search algorithm, this work provides the breadth necessary to draw more general conclusions about genetic algorithms' relative efficacy. It also provides some observations regarding the prevalent stochasticity of ABMs and its impact on search performance, and analyzes the utility of fitness caching for real-world problems. This rigorous set of experiments has been performed to obtain sufficient sample data to judge statistical significance, and as a result we can observe general trends and interesting patterns in the data. Generalizations are dangerous creatures, however, as Mark Twain mirthfully reminds us. I will leave it to the reader to decide whether all generalizations are indeed false, but the broader point stands that one must be cautious in applying them. Thus, this work presented in this chapter has no grandiose aspirations of being the last word on the subject. Rather, I view it as the opening sentence toward a healthy debate about search algorithm performance in this domain. Some may criticize my (necessarily constrained) choices of certain search parameters (e.g., the GA's mutation rate, or the cooling schedule for simulated annealing), and perform follow-up studies examining alternative choices. Others may discover that search performance differs on the specific model they are exploring, and wish to compare their ABM search task against the tasks described here. Arguably, the greater contribution of this chapter may be in developing a benchmark set of models and tasks to productively frame this

discussion, rather than in the specific numerical results obtained (although these results are themselves interesting). (The benchmark tasks, as concrete implementations of the QBME framework laid out in Chapter 3, are also valuable for demonstrating the breadth of model exploration tasks and how they can be successfully posed using a search-based paradigm.)

Finally, we take inspiration from an erudite epigram of Turing award-winner Alan Perlis [1982]: "Simplicity does not precede complexity, but follows it." So it is with the messy task of evaluating the performance of metaheuristic search algorithms. Only by wading through a sea of complex processes and data can we arrive at the far shore where we will discover simple guidelines and practicable solutions. One can stand forever on the shore, hoping the ocean will be shallow, the water will be warm, and that there aren't any sharks – but progress will never be made until we get our feet wet!

9.1. Description of Models and Tasks

This chapter presents a study of genetic algorithms with regard to their suitability for various tasks related to the development, exploration, and analysis of agent-based models. Specifically, we perform a comparative analysis of genetic algorithms against a baseline method (random search), as well as two comparable metaheuristic search techniques (random-mutation hill-climbing and simulated annealing). Because this is the first comparison of its kind, there is no standard set of tests or benchmarks in this domain against which search methods may be compared. We chose a range of models (simple, classic, and complex) and designed relevant model exploration tasks associated with each model. These "benchmark" tasks provide a basis on which the search methods will be judged. We selected models from the NetLogo sample models library, which is a large collection of agent-based

Model	**Description**	**Task**	**Dim.**
Fire	forest fire spread	find phase transition	1
Segregation	early social science ABM	find causes of segregation	2
Ants	ant food foraging	find most efficient foraging	2
Fireflies	synchronizing flashes	what encourages synchrony?	4
Flocking	flock/swarm motion	search for volatility	5
Daisy World	illustrates Gaia hypothesis	model error checking	6
Ethnocentrism	evolution of ethnocentrism	extreme scenario discovery & comparison	6
Heatbugs	abstract bio-inspired model	agent clustering/congregating	7
Wolf Sheep Predation	population dynamics	model calibration	9

Table 9.1. Benchmark ABMs and associated tasks chosen for evaluating search methods. The models are listed in increasing order of search space dimensionality (shown in the right-most column), which is equal to the number of free model parameters in the search task.

models ranging from simple example models to replications of published research models from various disciplines. The list of models/tasks is shown in Table 9.1.

These tasks were purposefully chosen to contain considerable variation in complexity and search difficulty. While search difficulty is challenging to quantify (*a priori*), one contributing factor is the dimensionality and size of the search space to be explored. In particular, the four earlier tasks (in Table 9.1) were chosen such that the search space was small enough to be fully enumerated (exhaustively explored). In other words, for these models it is possible to sample all combinations of settings of the free parameters with sufficient resolution (and with ample repeated sampling for statistical confidence) to obtain a "ground truth" map of the fitness landscape. This "ground truth" landscape will be useful for determining whether search processes have indeed found global optima, or whether they have become trapped in local optima (if they exist).

Despite their simplicity, these modeling tasks are not mere toy examples – they represent interesting questions about model behavior of real agent-based models. However, sophisticated search methods such as genetic algorithms are possibly unwarranted on these smaller search spaces; it is feasible to do a factorial-design enumeration of the space, which would give complete confidence in the results. Even so, having an efficient search mechanism to more quickly answer questions about the model in a heuristic manner is still useful, particularly for exploratory analysis of model behavior. Furthermore, these tasks provide a baseline that we would want any more sophisticated parameter search technique to handle without difficulty. Although the search spaces and dimensionality are small, some of the characteristics of the fitness landscapes encountered in these simple tasks may also be relevant to larger higher-dimensional parameter spaces. On the other hand, it may only be in higher dimensional spaces that a genetic algorithm will be able to take advantage of building blocks (composed of sets of parameters), and higher dimensional spaces are also more likely to have large numbers of local minima/maxima, so we must be careful about drawing general conclusions from these simple cases.

The later five (higher-dimensional) tasks involve search spaces that are too large to be enumerated in practice, so the true global optima is not necessarily known. However, these tasks are arguably more realistic in terms of the number of parameters that one might be searching in a typical ABM search task. Each of the search tasks will be described in greater detail.[1] The exact versions of the models used in the search experiments presented here are available for download from: `http://forrest.stonedahl.com/thesis/benchmark_models.zip`.

[1]In fact, the level of detail in the sections below borders on the tedious – I apologize for this in advance, but feel that little can be done to remedy this without sacrificing the scientific replicability of these experiments.

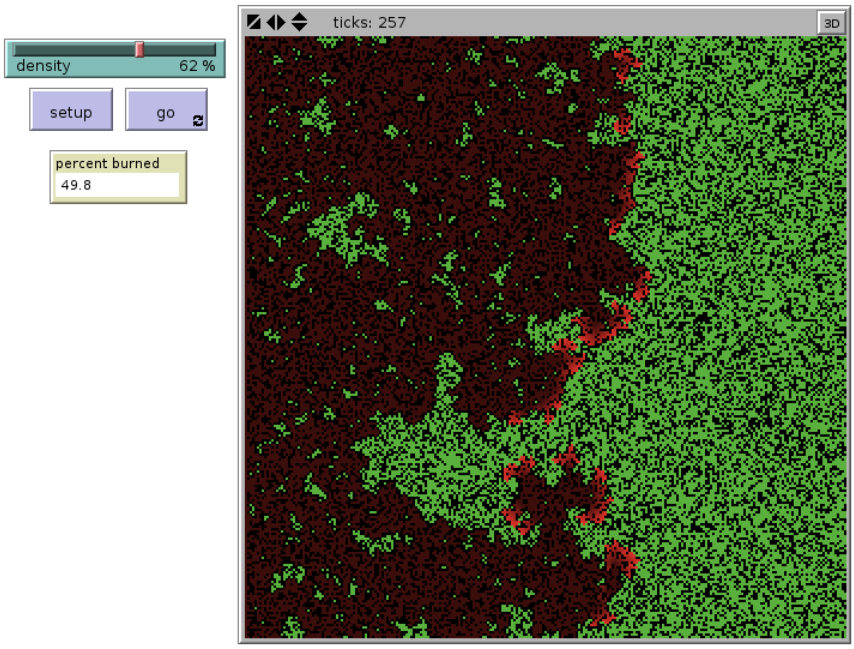


Figure 9.1. The NetLogo Fire model user interface.

9.1.1. Fire (FireVariance and FireDeriv)

The NetLogo Fire model [Wilensky, 1997c] (see Figure 9.1) demonstrates a critical point (or phase transition between two different behavioral regimes) in a simple percolation model. The Fire model has just one parameter density, which controls the density of the forest. While the simplicity of the model means that sophisticated algorithms are not necessary in order to detect this phase transition, it will provide a straightforward test case for searching for critical points, since the system's behavior is already well understood.

We actually performed two separate search tasks with the Fire model[2], though they both had the same common goal of identifying the critical threshold for the density parameter. For the *FireVariance* task, we searched for parameter settings that would maximize the standard

[2]Technical note: for our experiments the Fire model's world size was reduced from 251x251 to 99x99 patches, to improve efficiency. Qualitative model behavior was unaffected by this change. Unless stated otherwise, all tasks are based on the models from the NetLogo 4.1.2 model's library.

deviation of the amount of forest burned (across multiple runs with the same parameters. In other words, we're looking for parameters where sometimes only a few trees burn and sometimes the whole forest burns. For the *FireDeriv* task, we searched for parameter settings that maximized the discretely approximated derivative (a.k.a. *difference quotient*) of the average amount of forest burned with respect to the density parameter (Δdensity = 1.0). In other words, we were looking for parameters where a small change in density would cause a large change in the amount of forest burned. As mentioned in Chapter 3, these two approaches highlight different aspects of phase transitions, but either can be used to identify the phase transition in the Fire model. For both cases, the model is initialized (with the SETUP procedure) and run (the GO procedure) until there are no fires remaining, at which point we measure the percentage of the initial forest that burned. The density parameter is allowed to range between 1% and 99% by increments of 0.01 (for a search space composed of 9801 unique points). For both cases, the measures (derivative and variance-based) are averaged across some number of repeated model runs (see the various sampling amounts in Section 9.2.1 below) using the same parameter settings but with different pseudo-random number generator (PRNG) seeds.

9.1.2. Segregation

The NetLogo Segregation model [Wilensky, 1997d] (see Figure 9.2) is a replication of one of the earliest ABMs of how strong patterns of societal segregation may emerge from "weak discrimination" by individuals, based on the work of Thomas Schelling [Schelling, 1978]. The characteristic result of this model is that there is a disproportionate macro-level response to micro-level discriminatory practices. However, another surprising feature of this model is that having individual agents each seek maximal similarity (that is, being highly

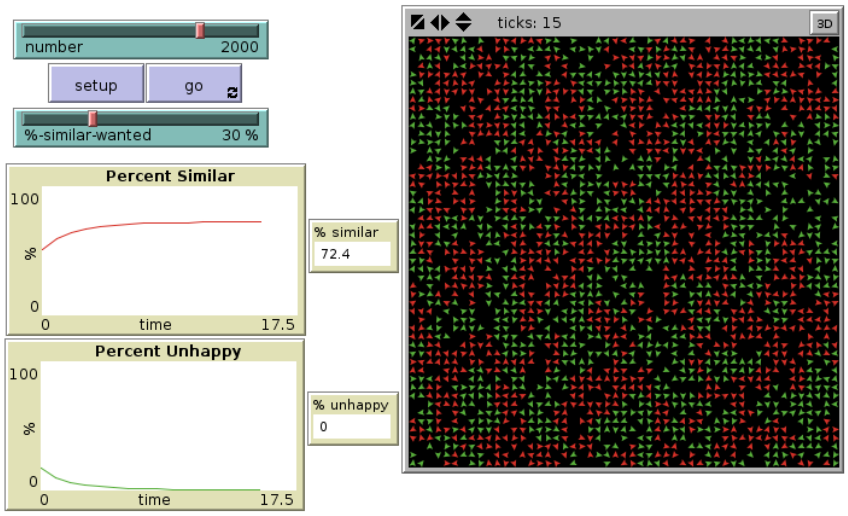


Figure 9.2. The NetLogo Segregation model user interface.

discriminatory) yields much less overall segregation than if agents are only moderately discriminatory. "Surprising" results like this are characteristic of the emergent behavior found in many agent-based models. The explanation, in this case, is that when agents are too strongly discriminatory (or "picky"), the model never settles down to a fixed equilibrium. Agents continually move from one location to another, almost always remaining unhappy with their surrounding neighborhood (unless the world population density is extremely low).

For this task, we searched for parameter settings that would result in maximal macro-level segregation by varying two parameters: the number of agents (from 500 to 1500 by increments of 10) and the percent-similar-wanted parameter (from 0 to 100 by increments of 1), which controls the micro-level discrimination. The complete search space thus contains 10,201 points, which is very manageable for enumeration. The model was initialized (SETUP), then run (GO) for 200 ticks (or until all agents were "happy" with their current locations), after which the global *percent-similar* was measured. This measure (again averaged across some number of repeated model runs) was maximized.

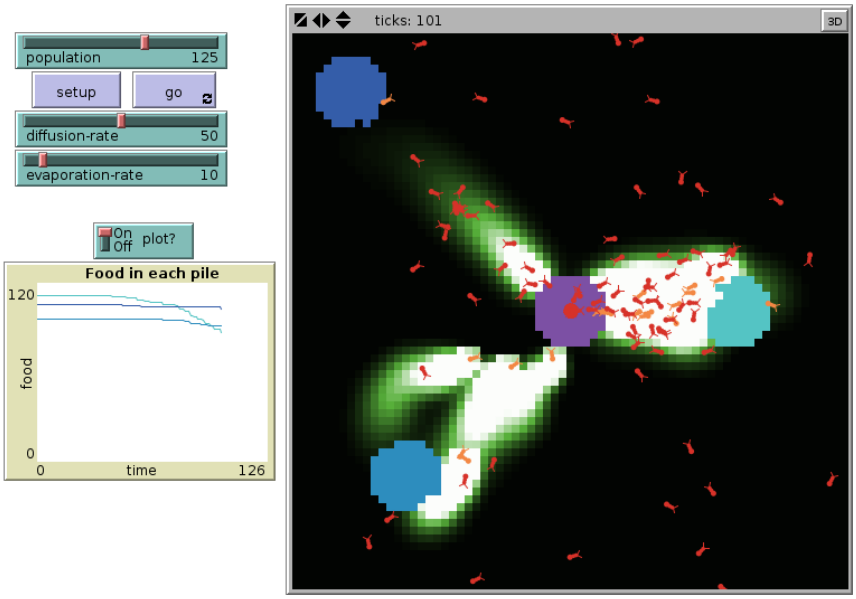


Figure 9.3. The NetLogo Ants model user interface.

9.1.3. Ants

The NetLogo Ants model [Wilensky, 1997a] (see Figure 9.3) is a pheromone-based ant foraging model, which was previously used by Calvez and Hutzler [2005] for a case study on using genetic algorithms for tuning parameters in agent-based models. Thus, this model is a logical choice to allow comparison with previous work – specifically this task will mimic Calvez and Hutzler's *Example 1*, which is about searching for parameters that yield the largest communal food harvest during a specified time period.

To compare results with Calvez and Hutzler, it was necessary to use an old version of the NetLogo Ants model, since the Ants model included in the NetLogo model's library was revised in 2005 to use a larger world grid and have the food deposits positioned further from the nest. This revision significantly changed the quantitative behavior of the model. Unfortunately, Calvez and Hutzler did not explicitly state which version of the NetLogo

Ants model they were using. Despite multiple attempts to contact the authors regarding a number of unspecified details in their paper (primarily about the details of the genetic algorithm they employed) I received no response. As a result, based on the publication date of their paper (2005), I estimated they had used NetLogo 2.1, which was the latest publicly released version of NetLogo in 2004. I also verified that there were no behavioral differences between the Ants model in NetLogo 2.1 and the previously released version (NetLogo 2.0.2) earlier in 2004, in case they had used that version instead. Furthermore, the NetLogo 2.1 version of the model qualitatively matches the model as depicted and described in Calvez and Hutzler's paper. Thus, I feel fairly confident that I am using the same model.

There are 3 explicit global parameters of this model (as well as several others implicit within the model code, which could be parameterized), but Calvez and Hutzler fixed one of them (the total number of ant agents) at the constant value of 10, thus allowing just 2 parameters to range freely during the search process. These two parameters are the diffusion-rate (ranging from 0 to 99 by increments of 0.1) and the evaporation-rate (also ranging from 0 to 99 by increments of 0.1). (The complete search space size is thus 982081 - almost 1 million combinations.) For this task, we measure the total amount of food harvested by the ants between 101 and 200 model ticks (inclusive), and we seek to maximize this measure (averaged across some number of repeated trials).

9.1.4. Fireflies

The NetLogo Fireflies model [Wilensky, 1997b] (see Figure 9.4) examines mechanisms by which fireflies might synchronize their flashing together, as exhibited in nature (e.g., *Pteroptyx cribellata*, *Luciola pupilla*, and *Pteroptyx malaccae* – particularly striking in the large

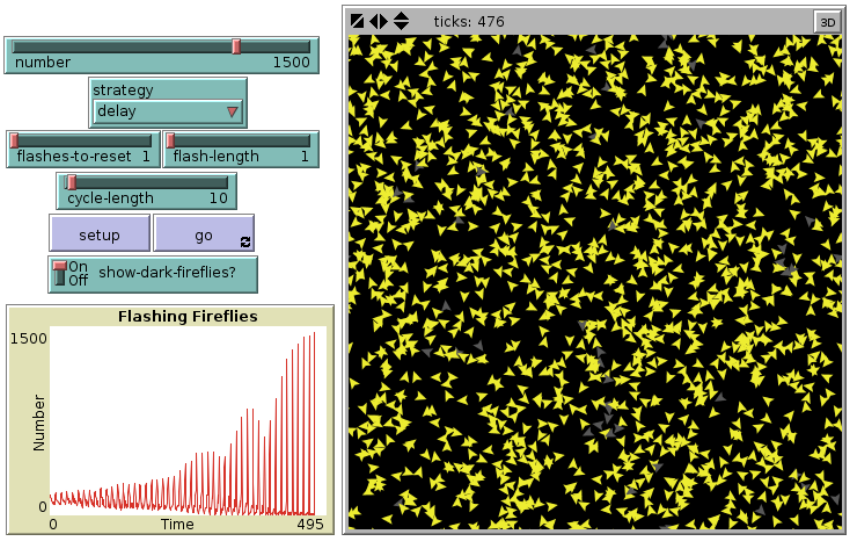


Figure 9.4. The NetLogo Fireflies model user interface.

firefly swarms of Southeast Asia). The Fireflies model is based in part on the research of Buck [1988] and Carlson and Copeland [1985].

In this model, there are two qualitatively distinct strategies (controlled by the strategy parameter) that the fireflies can use to try to synchronize with each other: *delay* and *advance*. For the most part, the *delay* strategy appears to consistently achieve global synchrony, whereas the *advance* strategy seems less effective, particularly at synchronizing the whole population. This is interesting, and begs the question: under what conditions is the *advance* strategy most effective at creating synchrony? Thus, our search task for the Fireflies model[3] fixes the strategy parameter to *advance*, and searches for parameters that lead to maximal synchrony. The number (of fireflies) parameter range goes from 10 to 50 in increments of 5, the flashes-to-reset ranges from 1 to 4, the cycle-length ranges from 5 to 20, and the flash-length ranges from 1 to 5. Although this task is searching a higher dimensional space than previous tasks, the size of the search space (only 2,880 combinations) is actually quite

[3]Again, for efficient experimentation we use a version of the model with a reduced world size.

small, due to the limited range and resolution for varying each parameter. For this task, we designed a special *synchrony-measure* (included in the downloadable benchmark model), which sums up the absolute difference (in modular arithmetic) between the internal clock states of each firefly with every other firefly, and then transforms and normalizes this value to be between 0 (unsynchronized) and 1 (completely synchronized). The model is initialized (SETUP) and run (GO) for 2000 ticks, and for each run the *median*[4] value of the *synchrony-measure* between 1900 and 2000 ticks is reported. This median synchrony measure is further averaged (using the mean, not median) across repeated model runs with the same parameter settings.

9.1.5. Flocking

As described earlier in Chapters 3 and 4, the NetLogo Flocking model [Wilensky, 1998] is based on Reynolds' classic "Boids" [C. W. Reynolds, 1987], and seeks to produce realistic-looking collective animal motion (such as flocking birds or schooling fish). This benchmark task is very similar (though not identical) to the tasks discussed in Chapter 4. In this case, we will be looking for flock volatility in the following sense: we are interested in what parameter settings would cause all of the birds to be simultaneously turning by large amounts at some point in the model (and we are curious to see what this pattern might look like). More specifically, during each tick, we are looking at the absolute heading change for each bird and choosing the minimum value across all birds. Thus, the turning measure is only registering the "weakest link" – if a single bird isn't turning at all, then the whole flock's volatility score for that tick is 0. However, we record the maximum of this measure across

[4]In some cases the *median* is a more appropriate measure of "average" or characteristic model behavior, since it is not influenced by outliers.

ticks (for 200 model ticks), so if there was ever a (simulated) moment when all of the birds were drastically turning, we will capture it. Finally we take the average value across some number of repeated model runs, and search to maximize this value. This task demonstrates one of the many combinations achievable by combining/condensing measures across different levels in different ways, as outlined in the QBME framework in Chapter 3.

The population parameter is fixed at 50 birds, but the five other model parameters are allowed to vary as follows. The vision parameter ranges from 0 to 10, the minimum-separation parameter from 0 to 5, and the max-align-turn, max-separate-turn, and max-cohere-turn parameters each from 0 to 20. In all cases, the increment of the range (resolution for searching the parameters) is 0.25. Note that the size of the search space here has skyrocketed to over 450 million parameter combinations, and enumeration of this space (at this resolution) is no longer a feasible strategy.

9.1.6. Daisyworld

The NetLogo Daisyworld model [Novak & Wilensky, 2006] (see Figure 9.5 is based on the model proposed by Watson and Lovelock [1983]. This model illustrates the Gaia hypothesis, and how Earth can be considered a single, self-regulating system including both living and non-living parts. Several years ago when examining this model, a colleague[5] and I discovered that there was a bug in its code, resulting in aberrant behavior for certain model parameter settings. As an example of an authentic ABM bug "in the wild", this model provided a natural opportunity to test the effectiveness of genetic algorithms for search-based anomaly detection, as a form of model checking or error testing. That is, could a GA-based search have picked up the same discontinuity in model behavior that we (humans) had discovered using

[5]Daniel Kornhauser

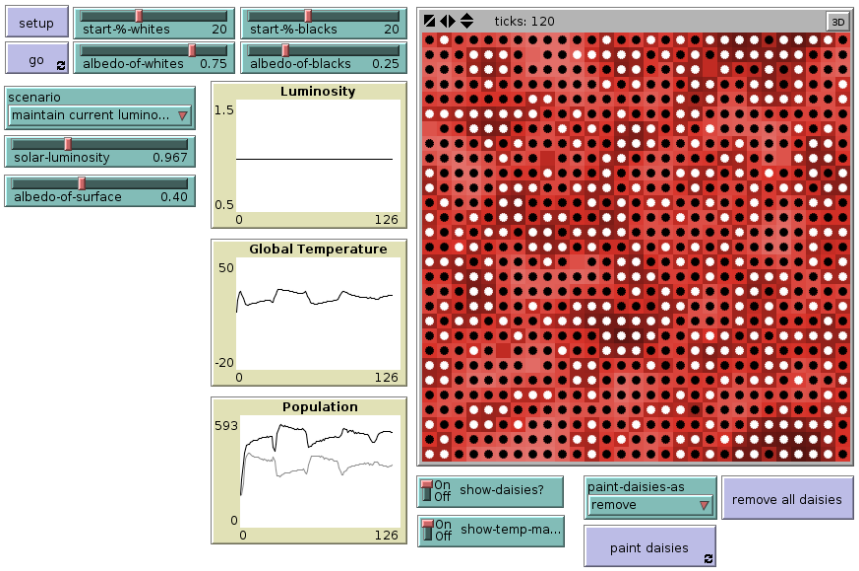


Figure 9.5. The NetLogo Daisyworld model user interface.

an interactive visual tool for plotting slices of the parameter space? Admittedly, the design of this task is *a posteriori*, since we already knew that a bug existed and the exploration measure was constructed such that it would be possible to find (although not trivial to locate, since the bug only manifested itself for a very narrow range of parameter settings).

For our exploration of the Daisyworld model[6], the anomaly we are trying to detect is a discontinuity in model behavior given a small change in an input parameter. Our exploration task is actually quite similar to the phase transition detection task employed for the Fire model. This overlap is fortunate, because it is plausible that model analysts looking for critical points and thresholds may at the same time uncover certain classes of model bugs/errors. In both situations, we are interested in maximizing the absolute change in some output measure (in this case we will use the total daisy population, although the discontinuity would be observable with many other output measures) given a small change in

[6]For this benchmark task we use the NetLogo 4.0.3 version of Daisyworld, as this is the version where the bug was discovered. Again, for search efficiency we reduced the world size to roughly 50% of that in the official Daisyworld model.

some parameter (in this case, the albedo-of-whites parameter). Specifically, we initialize the model (SETUP), run the model (GO), and measure the average (mean) number of daisies between 400 and 500 ticks. We do this for multiple repeated model runs and again take the average. We then calculate the discrete derivative of this value with respect to albedo-of-whites – that is, the change in the average number of daisies divided by the change in albedo-of-whites, where Δalbedo-of-whites is fixed at 0.01. The task is then to maximize the absolute value of this approximated derivative – thus finding those parameter settings where a small change in albedo-of-whites results in a large change in model behavior.

The parameter space to be searched includes 6 parameters: albedo-of-surface, albedo-of-whites, and albedo-of-blacks all range from 0 to 1 by increments of 0.01, start-pct-blacks and start-pct-whites range from 0 to 50, and solar-luminosity is allowed to range continuously[7]. The use of a continuous parameter makes it impossible to quantify this size of this search space in a way that is directly comparable to previous size calculations – however, the search space size when excluding that parameter is still over 26 million combinations.

9.1.7. Ethnocentrism

The NetLogo Ethnocentrism model [Wilensky & Rand, 2003] (previously pictured in Figure 1.1) is a classic model originally developed by Axelrod and Hammond [2003], demonstrating a possible mechanism by which ethnocentric behavior could arise as a result of locally played iterated prisoner dilemma games in an evolving population. This model is a good choice in part because its behavior has been carefully analyzed, and because particular effort went into

[7]Technically this "continuous" range is discretized by the precision of the computer's representation of floating point numbers, but this resolution provides an indistinguishable approximation of the continuous range, for our purposes here.

making sure that this NetLogo model was an accurate replication of Axelrod and Hammond's original model [Wilensky & Rand, 2007].

The main result from the Ethnocentrism model was that for a wide variety of parameter settings, "ethnocentric" behavior naturally evolves and dominates the alternative "altruist", "egoist", and "cosmopolitan" strategies in the population. A little experimentation reveals that *egoist* behavior can also be easily achieved in the model by changing the payoff structure so that the cost of giving is greater than the benefit of receiving; however, it is not clear what parameter settings (if any) might induce largely *altruistic* populations. Thus, for our exploration of this model[8], we will seek model parameters that are most favorable to *altruists*. This particular *extreme scenario discovery* task is closely related to *model sensitivity analysis*. Since the major result of the model is the dominance of ethnocentric behavior, finding parameters where an alternative strategy is surprisingly dominant (or at least highly effective) may provide insight about model robustness.

For this task, we fix two model parameters, immigrant-chance-cooperate-with-same and immigrant-chance-cooperate-with-different, at 0.5, to ensure there is no particular bias of new agents toward one of the four strategies.[9] We allow 6 model parameters to vary: immigrants-per-day ranges from 0 to 10, mutation-rate ranges from 0 to 1 in increments of 0.001, and the cost-of-giving, gain-of-receiving, death-rate, and initial-ptr all range from 0 to 1 in increments of 0.01. Our specific measure is the average (mean) fraction of the population that is altruistic between 200 and 300 model ticks.

[8]Again, for efficiency we used a version of the model with world-size reduced by roughly 50%.

[9]If we allowed these to vary, then the GA search would give us the unsurprising (and relatively uninteresting) result that "if you only allow new (immigrant) agents to be altruists, and you have a high immigration rate and low mutation rate, then the population will be highly altruistic!"

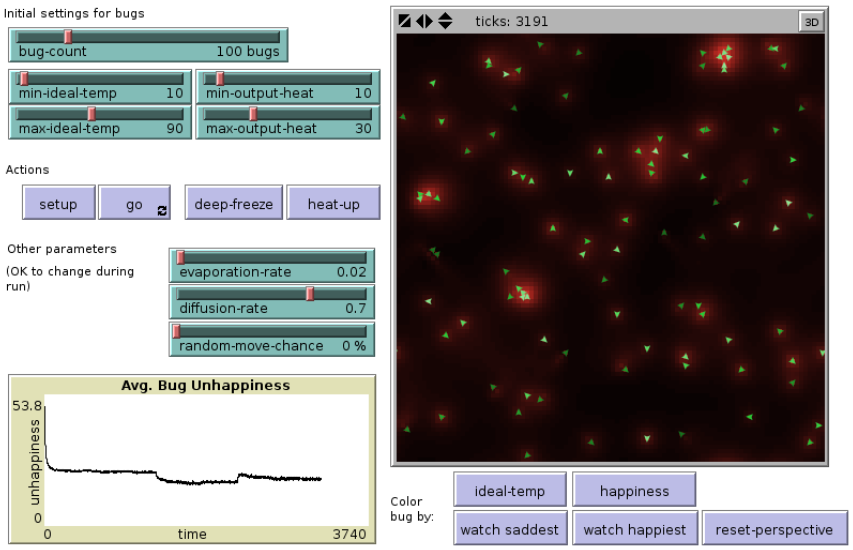


Figure 9.6. The NetLogo Heatbugs model user interface.

9.1.8. Heatbugs

The NetLogo Heatbugs model [Wilensky, 2004] has become a classic ABM of abstract emergent behavior, having been implemented in most major ABM toolkits since Swarm [Minar et al., 1996]. Individual bugs locally add heat to the environment, but also have temperature thresholds in order to be happy, and they will move if they are either too cold or too warm. However, each bugs movement and heat dispersal, as well as the global heat-diffusion rate, affects the other bugs in the environment. Running the model with typical parameter settings tends to result in a large number of small scattered bug clusters.

For this task, we are interested in discovering whether the Heabugs model[10] can exhibit extreme spatial clustering/congregating of agents. Specifically, we initialize the model (SETUP), run it (GO) for 1000 model ticks, and then calculate the average distance from

[10]Again, we used a reduced-world-size version of the model. We also transformed the original model's max-ideal-temp and max-output-heat parameters into ideal-temp-range and output-heat-range parameters. This transformation does not affect model behavior, and it avoids the problem of randomly-generated parameter settings having a larger min-ideal-temp than max-ideal-temp.

each bug to all other bugs. We then seek to minimize this measure – thus, the highest fitness behavior would be if every single bug was identically co-located. We hold the number of agents constant (bug-count = 25), while allowing 7 other model parameters to vary: evaporation-rate and diffusion-rate each range from 0 to 1 in increments of 0.01, min-ideal-temp ranges from 0 to 200 (integers), and ideal-temp-range, min-output-heat, output-heat-range, and random-move-chance all range from 0 to 100 (integers).

9.1.9. Wolf Sheep Predation

The NetLogo Wolf Sheep Predation model [Wilensky, 1997e] (introduced previously in Chapter 3 – see Figure 3.2) is a fairly simple simulation of predator-prey interaction in a closed ecosystem. This model demonstrates the oscillating dynamics that can result from food chain relationships. We will use this model for a calibration task, to determine how well it can exhibit a specific reference pattern. For a reference pattern, I chose the historical predator-prey dynamics of wolves and moose on Isle Royale in Michigan, U.S.A. [Vucetich & Peterson, 2009; Peterson, Page, & Dodge, 1984]. Isle Royale has been the site of an intense ongoing research effort since 1958, tracking the habits, numbers, and patterns of wolves and moose that live in this (essentially) closed ecosystem. As a result, over 50 years of population (abundance) data is publicly available on the Wolves and Moose of Isle Royale project website [Vucetich et al., 2011]. Figure 9.7 shows a plot of this data.

Admittedly, the Wolf Sheep Predation model is an abstract model of predator-prey dynamics and was not specifically designed for modeling the Isle Royale scenario, these two species, or a host of factors (such as disease) which affected the population dynamics of the real world data. Even so, I believe it is interesting to compare how well the patterns produced by this abstract model can match up with reality. Furthermore, even if you do not accept

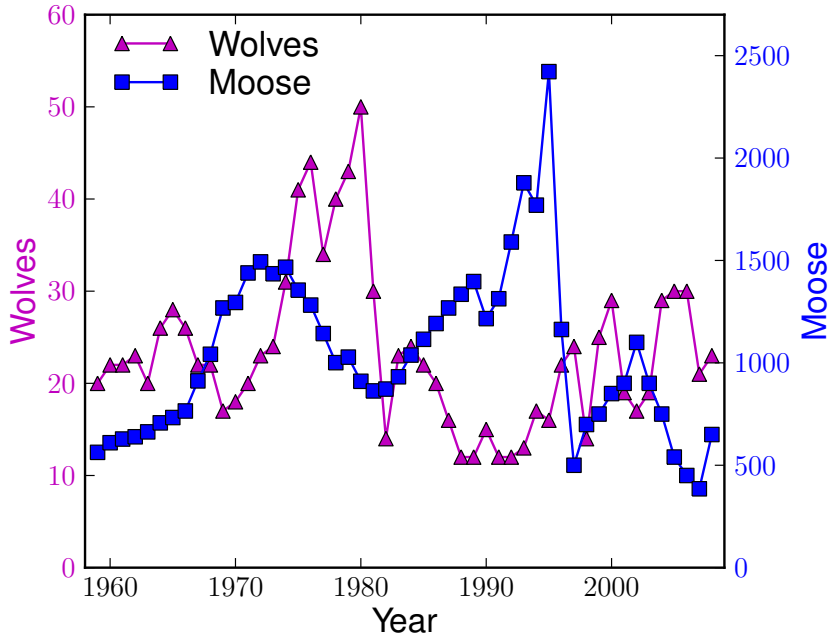


Figure 9.7. Wolf and moose abundances recorded on Isle Royale (Lake Superior) in Michigan, U.S.A. Source: *The Wolves and Moose of Isle Royale* project [Vucetich et al., 2011].

that comparison with this specific data is relevant, this type of calibration task is certainly important for ABM analysis, and can thus serve as a useful benchmark for performance (which is what we are using it for here).

One issue for this calibration task is that the model does not have any concrete time scale associated with it, whereas the reference pattern data is indexed by year. Since it is not at all clear how many model ticks should be equivalent to one real year, we introduced an additional model parameter named ticks-per-year. In doing so, we allow the search process itself to determine an appropriate time scale conversion that provides the best match with the data.

To measure degree of calibration, we measure Pearson's product-moment correlation coefficient between the simulated wolf population history (measured every TICKS-PER-YEAR ticks) and the real wolf population history, and also between the simulated sheep population and the real moose population. These two correlation coefficients (which may range between -1 and 1) are summed together, and we seek to maximize this value – thus perfect correlation with both data sets would yield a calibration score of 2. Note that the use of correlation coefficient provides more flexibility to the match, as opposed to requiring the model to match the exact numbers of wolves and moose. (This flexibility can also be interpreted as letting one wolf in the model stand for X real wolves, and one sheep in the model stand for Y real moose.) After all, we are more interested in the qualitative pattern than in facsimile reproduction of the history.

There was another technical issue with the Wolf Sheep predation model, which is that under certain parameter settings this model can produce exponential sheep growth that can quickly consume all of the computer's RAM and possibly cause the process to hang. Since these explosive patterns are never going to be a good match with the real data anyway, we chose to stop the simulation and return a bad fitness score (-2) whenever the sheep population crossed a threshold of 10,000. While these technical details may seem minor, they are the typical sort of little issues that must be resolved to successfully perform search-based ABM exploration and analysis.

For this task we varied all 9 of the model parameters (excepting parameters like show-energy, which only affect visualization of the model). The boolean parameter grass? was allowed to be either TRUE or FALSE. All other parameters were integers: the initial populations (initial-number-sheep and initial-number-wolves) ranged from 1 to 250, grass-regrowth-time and wolf-gain-from-food both ranged from 0 to 100, sheep-gain-from-food ranged from

0 to 50, and wolf-reproduce ranged from 0 to 20, sheep-reproduce ranged from 1 to 20, and ticks-per-year ranged from 1 to 20. The total size of this search space is around 5.4×10^{12} parameter combinations.

9.2. Experimental Setup

9.2.1. Search Method Variations

For each of the tasks above, five different search algorithms were applied: two genetic algorithm variants, and three other metaheuristic search algorithms for comparison. These are described

Random Search (RS). Random Search (RS) simply chooses a point in the search space uniformly at random and evaluates its fitness, and determines whether it is better than any previously examined point. At the end of the search, the best examined point is returned.

Random-Mutation Hill Climber (HC). The Random-Mutation Hill Climber (HC) starts at a random point in the search space, consider it's *current location* in the space. It generates a *neighboring location* by applying the mutation operator (mutation-rate = 0.05) to the *current location*. It examines the fitness of the *neighboring location*, and if the fitness is superior, that *neighboring location* becomes the *current location*, and the process repeats. In addition, the HC has a mechanism to restart if it is "stalled". Specifically, if the HC makes 1000 attempts to move to a neighboring point without succeeding, the *current location* of the HC will jump to a new random location in the search space, and continue "climbing" from there. At the end of the search, the best examined point is returned.

Simulated Annealing (SA). Simulated Annealing (SA) [Kirkpatrick, Gelatt, & Vecchi, 1983; Černỳ, 1985] is a nature-inspired metaheuristic search algorithm based on physics (rather than biology, like genetic algorithms). The analogy here is to the metallurgical

annealing process, wherein a material is first heated, and then cooled in a controlled process in order to form larger purer crystals. Higher temperatures permit more random motion of the atoms through various (higher energy) states, and the slow cooling improves the likelihood that the material will converge into a lower energy configuration. In SA, this corresponds to initially allowing the search process to wander through poor fitness regions, but over time decrease the probability of moving to poor fitness regions so that it will eventually settle into a very good fitness region. Like the GA, the probabilistic nature of the SA sometimes allows it to avoid becoming trapped in local optima of the fitness landscape.

The SA algorithm itself is strikingly similar to the HC algorithm presented above. The SA starts at a current location, and it moves to (randomly generated) neighboring location if that location is superior, or possibly even if that location is inferior (with some *acceptance probability* based on how much worse the new location is and on the global "temperature" T). The probability of accepting an inferior position decreases over time as the global temperature T is cooled according to an "annealing schedule". Specifically, for these experiments the acceptance probability for SA was $e^{(\Delta fitness)/T}$, and we used a fairly standard exponential decay annealing schedule for decreasing temperature (specifically, $T = 0.99^t$, where t is the number of "moves" the SA has completed). In an ideal world, we would be able to vary the search parameters (such as the cooling schedule and mutation rate) to find optimal settings for the benchmark set, or for each model individually, but pragmatic constraints on time and effort are preventative. However, this lack of parameter tuning is in a sense "fair" across the search methods being compared - in all cases (including the GAs) search parameters were configured (a priori) to what seemed reasonable defaults, either based on standard best practices, previous literature, or intuition. A more detailed future investigation of this area

is desirable, but the current experimental design already required significant computational effort (see Section 9.2.4 below).

Generational Genetic Algorithm (GA-Gen). The generational genetic algorithm (GA-Gen) is based loosely on the simple genetic algorithm as originally proposed by Holland [J. Holland, 1975], though adapted for the more flexible chromosomal representation described in Section 9.2.2 below. The general idea of the generational GA (starting with a random population of individuals which evolves better solutions, generation by generation, as a result of the combined forces of variation and selection) has already been described in 3.4.1. All that remains is to fill in the specific details used in the GA-Gen implementation for this set of experiments.

Matching the HC and SA algorithms, the mutation-rate was set at 0.05. This 5% mutation rate may seem high compared to the mutation rates commonly employed with binary genetic algorithms (more commonly 1%, or inversely proportional to twice the length of the bit string). However, those smaller values were *per bit* mutation rates, whereas the 0.05% mutation rate described here is applied per-parameter, and there are only between 1 and 9 parameters being varied for these tasks, and numeric parameters are manipulated by Gaussian mutation, which is more likely to cause a small change in value rather than a large one. Thus, the amount of *novelty* introduced by a 5% mutation rate using this representation may be comparable to lower mutation rates using a strictly binary genotype representation. The GA's crossover-rate was set to 0.70 (which is a common/standard choice).

GA-Gen uses *tournament selection* (with tournament size 2) to select individuals to become parents for reproduction. In contrast to the original *roulette selection* which selects individuals proportionally according to their numeric fitness values, *tournament selection* examines two individuals from the population random and selects the more fit of the two.

Thus, only relative fitness comparisons matter and absolute fitness values are unimportant; this can save human effort that would be required to tailor and properly scale the fitness function for use with *roulette selection* on a given problem.

Steady State Genetic Algorithm (GA-SS). It has been argued that a more incremental population replacement strategy (as opposed to generational replacement) is beneficial for genetic algorithms to converge more quickly on optimal solutions [Whitley & Kauth, 1988; Whitley, 1989]. To test this claim in the context of ABM exploration, we also tried a GA using a "steady state" population replacement strategy. Our steady state genetic algorithm (GA-SS) is identical to the generational GA described in most respects. However, unlike the generational GA, the steady state only replaces a single individual in the population at a time. That is, it generates one new offspring (using the same parent selection, crossover, and mutation mechanisms), and then replaces one individual in the population (not necessarily the parent) with this new offspring. For these experiments, we always replace a member of the existing population chosen uniformly at random. (An alternative variant of the steady-state GA involves always replacing the individual with the worst fitness, which we did not investigate here, but is also supported by the *BehaviorSearch* tool.)

Intuitively, this incremental replacement can speed up the maximal rate of evolution, since a new good individual that enters the population may be consequently chosen to be a parent more quickly than in the generational model, which requires that the whole population is replaced in a batch operation. However, a counter argument can be made that using a steady-state GA might result in premature population convergence and decreased genetic diversity – thus the benefit in practice is not necessarily clear.

9.2.2. Chromosomal representation

While some of the experiments in the case studies presented in earlier chapters used Gray binary encoding for the model parameters, all searches in the experiments presented here used the "MixedType" chromosomal representation. This representation has the most straightforward mapping between genotype and phenotype – that is, each gene represents one parameter (which may be integer-valued, real-valued, boolean, or categorical). Mutation is applied (with a certain probability) to each gene individually, where Gaussian mutation is used for numeric values, bit-flipping is used for boolean values, and random choice is for categorical values. For the genetic algorithms, crossover only splits the chromosome in between genes (no special real-valued crossover mechanisms are employed to do crossover on the intra-gene level). (A comparative study of the relative efficacy of different chromosomal representations is an important area for future work, but is beyond the scope of the current study, which is already investigating the effects of changing multiple variables.)

9.2.3. Caching and Noisy Sampling Variations

For each of the five search algorithms described in Section 9.2.1 above, we tested 5 different levels of sampling to reduce uncertainty in the fitness evaluations, and we tested all of these combinations with and without fitness caching. This work dovetails on the more theoretical work about sampling noisy fitness landscapes presented in Chapter 8. Here we are seeking the empirical answer to the questions:

(1) when is fitness caching beneficial in the presence of noise?

(2) what level of sampling is most effective for efficient exploration of the fitness landscape?

9.2.4. A note on computational effort

In total, the benchmark experiments for the models presented below cumulatively required running roughly 300 million (3.0×10^8) agent-based simulations and took more than 37000 CPU-hours[11] It is only with the advent of high performance cluster computing that these type of experiments have really become feasible. However, note that practitioners of this exploratory approach need not necessarily expend this much computational effort in their own ABM analysis work. For the purpose of benchmarking, 30 repeated searches with each search configuration were required, in order to have reasonable statistical confidence for comparing the performance of different search configurations. Furthermore, 50 different search configurations (caching vs. not, different noise sampling, 5 different search algorithms) were tested for each model. While it may be advisable for practitioners to try a few repeated searches (to reduce the danger of anomalous results) and use more than one search configuration (in case a specific search method yields particularly poor results for the problem), they should not require anywhere near 1500 searches per exploratory task in order to discover interesting parameters or answer relevant analysis questions. Furthermore, much of the power of the query-based exploration framework, is that once you have found a certain behavior, you have proof of the existence of that behavior in the model's parameter space. You may not know for certain that the behavior can't be achieved (or achieved more fully) using different parameters, but you do know that it is at least possible. Thus, even running just a single search (which can generally be run on any modern single/dual/quad-core laptop or desktop computer) has the potential to provide significant insight into your model's

[11]This equates to a little over 4 CPU-years, meaning that if a single processor had been responsible for this task, it would have spent nearly as long on this thesis as I have. Of course this effort was instead spread across hundreds of processors for a much briefer amount of time. (If only I could have similarly parallelized the human aspect of this thesis!)

analysis, if the result is positive. On the other hand, if the search result is negative, then many more searches would need to be attempted before deciding that the desired behavior (probably) cannot be elicited from the model.

However, it is the coming ubiquity of massively multi-core machines combined with the increasing prevalence of parallel computing clusters as well as elastic on-demand cloud computing that holds the most promise for the QBME methodology to gain popularity and become mainstream in the future. Currently these technologies are a few steps removed from the scientific practice, but eventually tools will arise to integrate with (and tap into) incredible computational resources. In the not-so-distant future I envision in a "begin parallel search" button appearing in toolkits like NetLogo that would seamlessly launch dozens or hundreds of simultaneous genetic algorithms searches on a remote grid/cluster, reporting back the most promising results to the user as they are discovered in real-time.

9.2.5. A note on measuring search performance

There are a number of ways to measure the performance of a search algorithm so we will clarify our method here. Each search algorithm (GA, HC, RS, etc) evaluates fitness of individual points in the search space as it progresses. As it runs, it keeps track of the best individual (and associated best fitness) discovered so far, and if it finds a new individual that's better than the previous best, it records that as the "best" instead. This history of best fitness values found, along with the number of evaluations (simulation model runs) that were required before the fitness was found, forms a natural way to chart the progress of the search.

However, because fitness evaluation is noisy, the fitness values reported by the search process are biased towards better fitness than the points actually represent. Consider, for

instance, a population of 100 individuals that each have a true average fitness of 10, but that noisy fitness evaluations causes them each to report $10 + R$, where R is a normally distributed random number with mean 0 and standard-deviation of 1. It's more likely than not that the Genetic Algorithm will evaluate one of the individuals and find a fitness greater than 12, even though the true fitness is only 10. For this reason, we employ a method we call *best-checking*[12], which is extrinsic to the search algorithm, but very useful for accurately assessing search performance. With *best-checking*, we take each new supposed "best" individual reported by the search method, and re-evaluates the fitness for that individual using an additional B model runs. This provides an *unbiased* estimate of the fitness of each individual that the search process views as better than all previous. As we discussed in Chapter 8, it is quite possible that the search process will choose a new individual which is not better than the previous best, although it may appear so due to noise. The *best-checking* procedure has the additional benefit that we may choose the number of unbiased sampling runs (B) to be much larger than the regular amount of sampling for evaluating fitness, since *best-checking* is only invoked each time the search algorithm finds a new "best" – which occurs relatively infrequently, especially in the latter portion of the search process. For all of the experiments presented here, we use $B = 100$ additional model runs to obtain an unbiased estimate of the true fitness of each alleged "best" individual. Thus, the *performance* versus *number of model runs* plots presented in this chapter are showing the average (across 30 searches) of the "checked" fitness values of the best individuals found by the time the search has performed x model runs. (For all of these tasks we are measuring the "offline" performance of the algorithm; measuring "online" performance does not make sense in this context [K. A. De Jong, 1975].)

[12]This functionality is included in the BehaviorSearch software.

9.3. Benchmark Results

The results are organized as follows. We will first provide a birds-eye-view of search performance across all benchmark tasks, then we will delve deeper into the details of search performance by examining the search dynamics for selected illustrative cases. Next we will address the efficacy of fitness caching in the noisy environments created by ABM exploration tasks, followed by a discussion of the impact of varying levels of noise reduction through repeated sampling.

9.3.1. Performance summary

Choosing the criteria on which to measure search performance is not a simple matter. In some cases researchers have used the "time taken to find the optimal solution". However, this criteria only makes sense if the optimal solution is known ahead of time, and if the search task demands that the optimal solution be found. For the situation of ABM parameter exploration, we can only find the optimal solution through exhaustive enumeration in very small search spaces, and in general, we are interested in achieving good performance on the search task, rather than demanding perfection. Thus, we are interested in how quickly the search algorithm can achieve good performance (where "good" is relative to the performance achieved by other search algorithms). This leaves us with two variables: search time (measured by the number of simulations the search has run) and search performance (measured by the fitness of the best parameters the search has discovered so far). When evaluating performance on a single problem, we can look at the full plot of search performance over time. This can tell us, for instance, whether a certain algorithm does better early on, but is eventually surpassed by a different algorithm (if the search is allowed to run long enough).

However, in this benchmark test, we have 10 model exploration tasks, each with 5 different levels of noise reduction sampling[13], and fitness caching turned on or off, and evaluating the performance in this fine-grained fashion would require examining 100 dense performance plots, each comparing the 5 search methods tested here. We will examine a few such plots in the detailed discussion of each model task in sections 9.3.2.1 to 9.3.2.8 below, but this is ineffective as a comprehensive overview. Thus, it is necessary to condense search performance information, so that it can be summarized and more easily digested. One common approach is to measure the search performance of each method at the end of the searches. This is a logical performance measure since it reflects the best search results found by each search algorithm, given the amount of time that it was allowed to run. This is the first measure we use to quantify search algorithm performance. However, the search time cutoff (20K model runs) was chosen somewhat arbitrarily – if a smaller limit had been set, a different search algorithm might have been superior at that earlier point. This is particularly relevant for problems where all the search methods reach a similar performance plateau, but some arrive there more quickly than others (e.g., for a dramatic case in point, see the *Segregation* task results discussed below in Section 9.3.2.2). In order to reward search algorithms for both the quality of the solutions *and* the speed in arriving at them, we also looked at a measure that averages the search performance across time. This is equivalent to the assumption that a search practitioner might have chosen to stop the search (with equal probability) at any point prior to our search cut-off of 20K model runs. This second measure is proportional to the area under the curves in the average search performance versus time plots. Any form of condensing/averaging data has its caveats, since some information is being lost. However,

[13]Except for the FireVariance task, which used only 4 levels of sampling, since a sample size of 1 does not permit the calculation of variance!

these two measures provide reasonable proxies for search algorithms' performance that is useful for the type of ABM exploration tasks we are interested in.

The condensed benchmark results using the first measure (performance at the end of the search) are shown in Table 9.2 (with fitness caching turned on) and Table 9.3 (without fitness caching). Similarly, the condensed benchmark using the second measure (average performance across search time) are shown in Tables 9.4 and 9.5. Even after this condensing of temporal search information (and averaging performance across 30 repeated searches), global trends in the benchmark performance can be somewhat difficult to detect. Since the best performance in each row is shown in bold, scanning down the tables, one may perceive that GA-Gen (the generational genetic algorithm) fairly often achieved the best performance of the five methods, and the larger number of bold entries lower in the table show that this was more true in the exploration tasks that had larger, more complicated, and higher dimensional search spaces.

However, to appropriately summarize the benchmark results, another level of information condensing is required. Because the performance/fitness values are incomparable across different modeling tasks, we cannot simply average performance values – instead we use a rank-order approach to aggregate data across tasks. Each row in these tables corresponds to a specific task and an associated noise level sampling. For each row, we rank the five search algorithms 1, 2, 3, 4, and 5, with 1 corresponding to the search algorithm that did the best under those conditions, and 5 corresponding to the search algorithm that did the worst. Then, for each search method, we can take the average of it's rank value over all the rows: e.g., a search algorithm that achieved rank 2 in half of the cases and rank 3 in the other half would get an average rank of 2.5. These average rank results are given in Table 9.6. Using either performance measure, with and without fitness caching, GA-Gen yielded

Model/Task	Sampling	RS	HC	SA	GA-Gen	GA-SS
FireDeriv	1	9.29	10.25	10.04	**10.53**	9.65
FireDeriv	4	13.41	12.86	**13.92**	13.17	13.59
FireDeriv	9	12.06	12.61	**13.84**	12.49	12.80
FireDeriv	16	13.53	11.80	13.80	**14.37**	12.68
FireDeriv	25	**13.46**	11.11	12.54	12.77	12.99
FireVariance	4	**15.40**	13.70	13.86	14.86	14.63
FireVariance	9	**18.35**	17.41	17.47	17.79	18.21
FireVariance	16	**20.04**	19.22	19.21	19.64	19.14
FireVariance	25	19.84	18.33	19.65	**20.28**	19.97
Segregation	1	99.00	99.59	**99.68**	99.52	99.28
Segregation	4	99.67	99.90	99.92	99.85	**99.95**
Segregation	9	99.92	99.90	99.94	99.94	**99.97**
Segregation	16	99.96	99.97	**99.97**	99.97	99.94
Segregation	25	99.96	**99.97**	99.97	99.94	99.93
Ants	1	22.55	20.77	22.12	**22.72**	21.89
Ants	4	24.38	24.61	**25.10**	24.73	24.35
Ants	9	25.18	25.52	24.32	**25.85**	25.37
Ants	16	25.74	22.70	25.91	**25.97**	25.41
Ants	25	25.69	23.00	21.53	**26.12**	26.07
Fireflies	1	0.733	0.724	0.721	**0.734**	0.714
Fireflies	4	0.768	**0.775**	0.763	0.761	0.735
Fireflies	9	0.779	0.783	**0.785**	0.779	0.764
Fireflies	16	0.782	0.783	**0.789**	0.758	0.771
Fireflies	25	**0.791**	0.788	0.781	0.787	0.766
Flocking	1	13.65	16.21	18.19	**18.94**	18.84
Flocking	4	17.92	17.97	19.45	19.65	**19.66**
Flocking	9	19.90	18.58	19.80	19.89	**19.92**
Flocking	16	19.75	19.33	**20.01**	19.84	19.97
Flocking	25	19.67	19.51	19.82	**19.98**	19.94
Daisyworld	1	5747	7304	**13490**	11852	10436
Daisyworld	4	**19766**	15521	16971	18679	15848
Daisyworld	9	**21341**	16801	16243	20311	18203
Daisyworld	16	20108	11889	14486	**20304**	18471
Daisyworld	25	**20270**	7918	11815	19945	17052
Ethnocentrism	1	0.335	0.337	0.321	**0.407**	0.384
Ethnocentrism	4	0.388	0.382	0.400	**0.416**	0.413
Ethnocentrism	9	0.390	0.402	0.416	**0.418**	0.415
Ethnocentrism	16	0.387	0.377	**0.421**	0.417	0.418
Ethnocentrism	25	0.386	0.398	0.407	**0.417**	0.414
Heatbugs	1	9.87	10.06	**7.09**	9.94	10.32
Heatbugs	4	9.41	10.29	**7.97**	8.09	9.74
Heatbugs	9	9.32	10.73	8.89	**7.89**	9.06
Heatbugs	16	10.06	11.47	9.86	**8.59**	9.18
Heatbugs	25	10.30	11.79	8.94	**8.52**	9.35
WolfSheep	1	0.890	0.934	**1.14**	0.977	0.807
WolfSheep	4	0.890	0.922	1.05	**1.09**	0.878
WolfSheep	9	0.835	0.816	**1.04**	0.974	0.970
WolfSheep	16	0.883	0.727	0.861	**0.986**	0.843
WolfSheep	25	0.871	0.698	0.773	**0.942**	0.840

Table 9.2. Benchmark search performance (at end of search – 20K model runs) with fitness caching turned **on**. For each task and noise sampling level (row), the best performance is shown in bold. Each data point is the average of 30 searches.

Model/Task	Sampling	RS	HC	SA	GA-Gen	GA-SS
FireDeriv	1	8.04	9.02	7.77	11.55	**13.61**
FireDeriv	4	12.61	12.76	**13.48**	12.79	13.10
FireDeriv	9	12.61	13.29	**13.84**	13.50	13.44
FireDeriv	16	**13.77**	9.14	13.09	11.87	12.66
FireDeriv	25	12.75	9.21	**13.66**	12.69	13.10
FireVariance	4	15.09	14.66	16.31	16.83	**19.00**
FireVariance	9	18.59	16.74	19.07	**19.65**	19.57
FireVariance	16	20.25	17.40	19.13	**20.32**	19.68
FireVariance	25	**20.12**	16.77	19.85	20.04	19.41
Segregation	1	**99.60**	99.42	99.00	99.51	99.30
Segregation	4	99.73	99.83	**99.93**	99.88	99.87
Segregation	9	99.77	99.93	99.88	99.92	**99.95**
Segregation	16	99.93	99.94	**99.97**	99.96	99.95
Segregation	25	99.96	**99.97**	99.91	99.95	99.95
Ants	1	22.23	22.38	22.48	24.86	**25.27**
Ants	4	25.00	24.20	24.45	25.31	**25.32**
Ants	9	25.50	23.95	23.94	**25.97**	25.71
Ants	16	25.60	22.62	24.97	25.79	**25.80**
Ants	25	**25.97**	18.69	22.76	25.90	25.78
Fireflies	1	0.715	**0.742**	0.728	0.737	0.737
Fireflies	4	0.765	0.755	**0.782**	0.772	0.769
Fireflies	9	**0.791**	0.758	0.783	0.785	0.752
Fireflies	16	0.787	0.757	0.774	**0.789**	0.764
Fireflies	25	**0.788**	0.718	0.778	0.770	0.762
Flocking	1	13.31	15.81	16.97	**19.17**	19.00
Flocking	4	18.80	18.76	19.73	19.33	**19.84**
Flocking	9	19.75	**19.86**	19.75	19.82	19.76
Flocking	16	19.81	19.95	19.92	**19.95**	19.77
Flocking	25	19.79	18.78	19.32	**19.96**	19.91
Daisyworld	1	7894	8662	11020	12404	**12612**
Daisyworld	4	**20490**	13849	16585	18090	15149
Daisyworld	9	**21329**	14959	13537	17967	18311
Daisyworld	16	**20705**	12938	8150	17788	17727
Daisyworld	25	**19914**	11725	9069	18330	15957
Ethnocentrism	1	0.318	0.347	0.343	**0.424**	0.421
Ethnocentrism	4	0.395	0.377	0.397	**0.424**	0.420
Ethnocentrism	9	0.384	0.393	**0.419**	0.415	0.410
Ethnocentrism	16	0.390	0.371	**0.421**	0.415	0.392
Ethnocentrism	25	0.384	0.363	**0.412**	0.403	0.386
Heatbugs	1	10.35	10.49	7.11	**5.34**	6.05
Heatbugs	4	9.23	11.04	8.16	**6.27**	8.38
Heatbugs	9	9.25	11.77	9.27	**7.43**	9.18
Heatbugs	16	10.06	11.88	10.21	**7.91**	9.05
Heatbugs	25	9.72	12.60	**9.18**	9.75	10.28
WolfSheep	1	0.920	0.996	**1.07**	0.947	0.803
WolfSheep	4	0.898	0.845	**1.04**	0.973	0.877
WolfSheep	9	0.898	0.675	0.897	**1.02**	0.882
WolfSheep	16	0.913	0.614	0.840	**0.964**	0.791
WolfSheep	25	0.838	0.590	0.770	**0.888**	0.744

Table 9.3. Benchmark search performance (at end of search – 20K model runs) with fitness caching turned **off**. For each task and noise sampling level (row), the best performance is shown in bold. Each data point is the average of 30 searches.

Model/Task	Sampling	RS	HC	SA	GA-Gen	GA-SS
FireDeriv	1	10.39	10.51	10.18	**10.72**	9.96
FireDeriv	4	13.12	11.97	**13.74**	12.61	13.36
FireDeriv	9	11.65	11.84	**13.60**	12.48	12.50
FireDeriv	16	12.61	10.03	13.06	**13.11**	12.24
FireDeriv	25	12.17	10.77	11.71	12.16	**12.32**
FireVariance	4	**15.90**	14.01	14.54	15.03	15.27
FireVariance	9	**18.49**	16.92	18.32	18.03	18.04
FireVariance	16	19.36	17.45	**19.61**	18.78	19.09
FireVariance	25	18.95	15.78	19.52	**19.82**	19.37
Segregation	1	99.00	99.58	**99.68**	99.52	99.28
Segregation	4	99.66	99.86	99.88	99.84	**99.95**
Segregation	9	99.90	99.78	99.87	99.90	**99.94**
Segregation	16	**99.91**	99.79	99.72	99.91	99.88
Segregation	25	**99.88**	99.78	99.62	99.87	99.84
Ants	1	22.44	21.07	22.32	**23.35**	22.13
Ants	4	24.47	23.28	23.98	**24.71**	24.48
Ants	9	25.33	22.85	22.10	**25.59**	25.34
Ants	16	25.71	20.87	24.38	**25.78**	25.40
Ants	25	25.66	20.50	20.64	**25.93**	25.63
Fireflies	1	0.732	0.723	0.720	**0.734**	0.714
Fireflies	4	0.765	**0.770**	0.760	0.760	0.733
Fireflies	9	0.775	**0.778**	0.771	0.775	0.760
Fireflies	16	0.770	**0.773**	0.757	0.752	0.767
Fireflies	25	0.770	0.772	0.744	**0.774**	0.760
Flocking	1	13.15	15.98	18.45	**18.92**	18.77
Flocking	4	18.72	18.11	19.38	**19.53**	19.48
Flocking	9	19.34	18.38	19.14	19.60	**19.64**
Flocking	16	19.15	18.15	18.57	19.38	**19.46**
Flocking	25	18.74	17.89	18.42	19.13	**19.15**
Daisyworld	1	5974	7212	**12066**	11541	9188
Daisyworld	4	**19212**	11817	13857	17964	15747
Daisyworld	9	17566	13071	12520	**18397**	16233
Daisyworld	16	16323	9035	9980	**17268**	15770
Daisyworld	25	14076	6362	9036	**15816**	14218
Ethnocentrism	1	0.336	0.342	0.339	**0.403**	0.382
Ethnocentrism	4	0.382	0.371	0.385	**0.409**	0.406
Ethnocentrism	9	0.378	0.365	0.387	**0.404**	0.401
Ethnocentrism	16	0.373	0.332	0.365	0.395	**0.399**
Ethnocentrism	25	0.371	0.353	0.330	**0.384**	0.383
Heatbugs	1	10.18	10.96	**8.98**	10.10	10.53
Heatbugs	4	10.15	11.76	9.70	**8.71**	9.96
Heatbugs	9	10.06	12.43	10.67	**8.93**	9.86
Heatbugs	16	10.60	12.85	11.56	**9.91**	10.26
Heatbugs	25	10.88	12.88	10.94	**10.07**	10.36
WolfSheep	1	0.745	0.715	**0.968**	0.901	0.765
WolfSheep	4	0.754	0.711	0.814	**1.00**	0.779
WolfSheep	9	0.749	0.617	0.760	**0.875**	0.859
WolfSheep	16	0.734	0.567	0.653	**0.850**	0.762
WolfSheep	25	0.737	0.495	0.556	**0.787**	0.739

Table 9.4. Benchmark search performance (averaged across time) with fitness caching turned **on**. For each task and noise sampling level (row), the best performance is shown in bold. Each data point is the average of 30 searches.

Model/Task	Sampling	RS	HC	SA	GA-Gen	GA-SS
FireDeriv	1	9.44	8.60	8.57	11.57	**12.54**
FireDeriv	4	12.42	12.19	**12.76**	12.30	12.72
FireDeriv	9	12.33	9.28	**13.26**	12.52	12.27
FireDeriv	16	**12.67**	9.19	12.35	11.39	11.54
FireDeriv	25	11.93	9.22	**12.46**	11.97	11.38
FireVariance	4	16.07	15.24	15.98	17.00	**18.13**
FireVariance	9	18.44	17.58	18.28	18.96	**18.98**
FireVariance	16	**19.44**	15.12	18.49	19.30	19.20
FireVariance	25	**19.74**	16.61	19.29	19.50	18.55
Segregation	1	**99.60**	99.41	99.00	99.51	99.31
Segregation	4	99.72	99.75	99.85	**99.87**	99.86
Segregation	9	99.85	99.81	99.75	99.86	**99.91**
Segregation	16	**99.88**	99.61	99.72	99.88	99.88
Segregation	25	99.87	99.71	99.69	99.84	**99.88**
Ants	1	22.08	22.25	21.81	24.96	**25.04**
Ants	4	24.97	21.67	23.27	25.12	**25.24**
Ants	9	25.21	20.53	21.73	**25.85**	25.57
Ants	16	25.45	20.04	21.45	25.41	**25.51**
Ants	25	**25.77**	18.52	21.89	25.57	25.36
Fireflies	1	0.721	0.715	**0.741**	0.737	0.733
Fireflies	4	0.766	0.732	0.771	**0.773**	0.760
Fireflies	9	**0.777**	0.728	0.751	0.772	0.743
Fireflies	16	**0.778**	0.730	0.756	0.777	0.749
Fireflies	25	**0.770**	0.709	0.749	0.756	0.754
Flocking	1	13.17	15.88	17.24	**19.13**	18.96
Flocking	4	19.12	17.92	19.37	19.13	**19.59**
Flocking	9	**19.45**	18.24	19.21	19.22	19.40
Flocking	16	**19.23**	19.17	18.87	19.22	18.86
Flocking	25	**19.01**	17.50	17.66	18.96	18.91
Daisyworld	1	6688	7239	8448	11870	**12136**
Daisyworld	4	**18689**	10367	11151	17162	13922
Daisyworld	9	**18421**	9495	8384	14738	15511
Daisyworld	16	**15672**	10910	7390	14586	14317
Daisyworld	25	**15495**	9717	8046	14398	12642
Ethnocentrism	1	0.333	0.347	0.352	**0.416**	0.409
Ethnocentrism	4	0.385	0.348	0.386	**0.406**	0.403
Ethnocentrism	9	0.373	0.352	0.383	**0.395**	0.388
Ethnocentrism	16	0.376	0.351	0.366	**0.385**	0.375
Ethnocentrism	25	0.367	0.343	0.339	**0.373**	0.363
Heatbugs	1	10.74	11.74	9.20	**7.66**	8.68
Heatbugs	4	9.89	12.48	9.60	**8.36**	9.59
Heatbugs	9	10.05	12.80	10.61	**9.00**	10.32
Heatbugs	16	10.75	13.14	11.66	**9.96**	10.07
Heatbugs	25	**10.69**	13.03	10.88	10.71	11.00
WolfSheep	1	0.793	0.677	**0.904**	0.813	0.739
WolfSheep	4	0.777	0.640	0.827	**0.865**	0.761
WolfSheep	9	0.738	0.426	0.691	**0.890**	0.785
WolfSheep	16	**0.800**	0.429	0.658	0.782	0.702
WolfSheep	25	**0.750**	0.466	0.542	0.727	0.664

Table 9.5. Benchmark search performance (averaged across time) with fitness caching turned **off**. For each task and noise sampling level (row), the best performance is shown in bold. Each data point is the average of 30 searches.

Model/Task	Sampling	RS	HC	SA	GA-Gen	GA-SS
End-of-search performance	Caching	3.2	3.9	2.7	**2.0**	3.2
End-of-search performance	No Caching	3.2	4.2	3.0	**2.0**	2.7
Avg-over-time performance	Caching	3.0	4.3	3.3	**1.8**	2.6
Avg-over-time performance	No Caching	2.4	4.7	3.5	**1.9**	2.6

Table 9.6. Average performance *rank* for each of the search methods. The possible range of rank values is between 5.0 and 1.0, with lower scores being superior. This table shows average ranks (1-best, 5-worst) for the search algorithms across all exploration tasks and noise levels. The best average rank values for each case are shown in bold.

the best performance in all cases. This confirms the trends that were mildly apparent in the long-form benchmark results data (Tables 9.2 through 9.5). However, the GA-Gen average rank is around 2, indicating that although it often was the best search algorithm (rank 1), for many cases another search algorithm provided better performance. Again, this summarizes a trend that is evident the long-form benchmark results. While it would have been nice to find that one search method consistently dominated all others, the truth (as it often happens) is more subtle and complicated. In fact, there are a few surprising features of these benchmark results.

For example, the random search (RS) method generally outperformed the hill climbing (HC) heuristic. And when looking at the average search performance over time (which also considers early search performance), RS also has a better average rank than SA and GA-SS. RS was chosen as a baseline unintelligent uniform sampling procedure - and yet it provides better performance than more sophisticated search methods in many cases. How can this be? There are a number of contributing factors. First, many of these tasks have a relatively small search space, and the chances of randomly sampling a good solution are relatively high. All of the other search methods (but especially HC and SA) have some notion of moving incrementally through the search space. Thus, if they start by sampling a poor fitness region

of the space, it takes some time for them to climb out of it into a better fitness region. HC (and to some extent SA) specialize in fine-tuning a solution, and this thrust toward "exploitation" over initial "exploration" can be costly for performance, especially early on in the search process. Genetic algorithm's population-based approach is less prone to this problem, since the initial population contains a decently large random sampling. Second, for many of these tasks noisy fitness is a significant issue – particularly when the noise sampling level is small (1, 4, 9). When noise is significant, HC (and other search techniques) may fail to pick up the appropriate search gradient, and not climb directly uphill. Instead, HC may wander (aimlessly) in a small region of the space, tossed back and forth by noisy fitness values. Of course, RS also experience noisy fitness evaluations, and this may mislead it in its estimation of the value of each set of parameters that it tests, but at least it does not become stranded in a small region of the space, where fitness may be low. In larger search spaces (higher dimensional, more ABM parameters, etc), where good fitness values are more sparse, and where fitness uncertainty does not overpower the search gradient, then one would expect HC to outperform RS. It is also possible that the 20K model run cut-off for the search was often too low for HC to excel on the tasks – if HC had been allowed to run for a much longer time, it may have eventually surpassed random search, after its fine-tuning capabilities became more beneficial. However, it is important to recall that many ABM analysis tasks require more exploration, and less exploitation/optimization, and thus RS may be a useful method, and should not be written off entirely. That said, GA-Gen, which uses a balance of exploration and exploitation, outperformed RS in most cases.

A second surprising result was the GA-SS generally performed worse than GA-Gen on these tasks. We explicitly included GA-SS as a search method, since previous work [Whitley & Kauth, 1988; Whitley, 1989] had suggested that GA-SS might be superior in its ability to

more quickly converge on good solutions. However, quick convergence can be a drawback as well as a benefit. It is likely that the GA-SS population prematurely converged to moderately good fitness values, possibly at local optima (either intrinsic to the fitness landscape or caused by noisy fitness evaluations), leaving little diversity in the population. After convergence to a near-identical population, further search progress would be slow, likely similar to the HC. On the other hand, GA-Gen converges more slowly, allowing the search to explore several branches in parallel, before one branch becomes dominant and the population converges. We should keep in mind, though, that these results presented here are not definitive, and it would be foolish of us to conclude that GA-Gen is superior to GA-SS for ABM exploration tasks. A more tempered conclusion would be that for these search spaces, which are somewhat conducive to random search, greater emphasis should be placed on exploration, and slower convergence may be more beneficial for this task. Furthermore, there are a number of methods for slowing GA population convergence, including increasing the mutation rate, increasing the population size, or more explicit diversity maintenance mechanisms. Thus, it could be that using different conditions than we used here, GA-SS might outperform GA-Gen on these same benchmark tasks. However, until more in-depth investigations can be carried out to assess the relative contributions of the differing GA population replacement models, our benchmark results currently suggest that practitioners should choose a generational approach.

9.3.2. Individual tasks and search dynamics

9.3.2.1. Fire (*FireVariance* and *FireDeriv*). While high dimensional search spaces are difficult to visualize, for the Fire exploration tasks our search space is 1-dimensional, so we will take the opportunity to examine it. A "ground truth" map of the search space

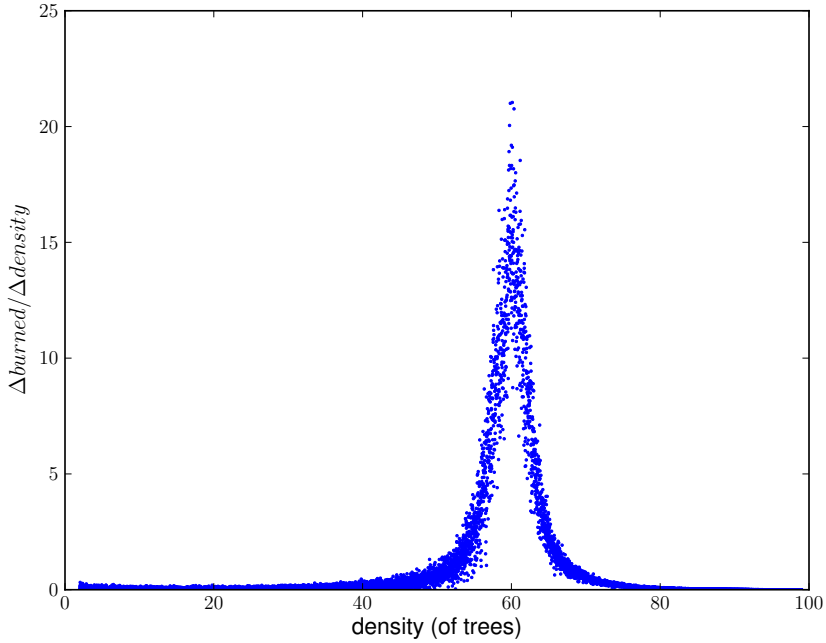


Figure 9.8. Fitness landscape for the *FireDeriv* task.

was created by using an exhaustive search to run the simulation with each possible density parameter value 100 times. This data was used to calculate the change in average percent-burned (per one unit change in density) at each density value (the "true" fitness landscape for the *FireDeriv* task, shown in Figure 9.8), and the standard deviation of the percent-burned at each density (the "true" fitness landscape for the *FireVariance* task, shown in Figure 9.9).

Note that despite taking 100 samples to reduce noise/uncertainty, the data is still somewhat noisy. The search algorithms were using smaller sampling values (1,4,9,16,25) than 100, meaning that their view of the search space was even more noisy than that displayed. In the exhaustive search, the highest fitness value for the *FireDeriv* landscape was found at

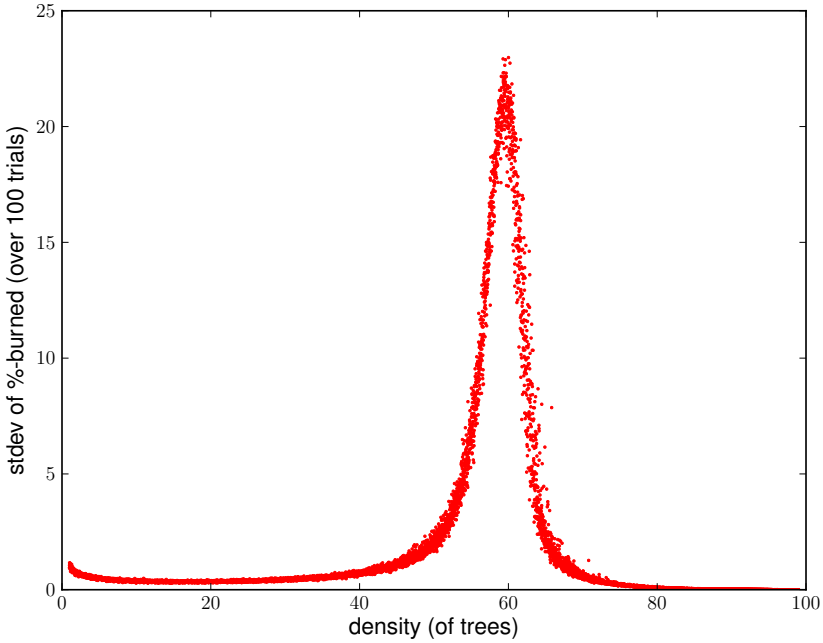


Figure 9.9. Fitness landscape for the *FireVariance* task.

density= 60.12.The highest fitness value for the *FireVariance* landscape was found at density= 60.01. Although these two density values are not identical, they are very close, and they both serve to identify the Fire model's phase transition. Although I have described the plots shown in Figures 9.8 and 9.9 as "fitness landscapes", the search space is also shaped by the search operators that navigate the space. For this task which has only a single parameter, the genetic recombination operator is essentially inoperative, and the only search operator that affects which points in the space are accessible from each other is the mutation operator. As mentioned earlier, these searches all use Gaussian mutation for numeric model parameters (such as density). Thus, a mutation can occasionally result in a large jump (i.e. large change in density), but small incremental steps are far more likely.

Apart from the noise, which may mislead or confound the search algorithms, both of these search spaces are fairly straightforward. They each have a global optimum that has a large basin of attraction leading up to it from either side. If you look carefully, you can also find a very small local optima near density= 0 (more apparent in Figure 9.9) - however its fitness is low and its basin of attraction is small, so searches are unlikely to be trapped by it. This local optimum is explained by chance and small numbers – for very low density values (e.g., 0.01 to 0.05), there may be only a few trees in the world, and if several of these trees happen to be created adjacent to the left border of the world, where the forest fire starts, then a large *percentage* of the forest can burn, simply because the "forest" consisted of only a few trees – not because the forest fire spread a great distance through the world.

A closer examination of the *FireDeriv* search results shows that a few searches (34 out of the 1500 searches performed for this task) were trapped by this inferior local optima. Of these 34, 5 happened using RS with Sampling=1 (both with and without caching), and the rest happened to HC with various sampling levels, although more often with higher sampling levels. In the worst case, using HC with Sampling=25 and no fitness caching, the search got trapped at the local optima 9 times out of the 30 searches, or 30% of the time. RS mistakenly chose the wrong optima because with Sampling=1, it was possible for a single lucky initial condition to achieve better fitness here (going between 0% burned at density= 1.01 to 100% burned at density=0.01) than was achievable in the primary phase transition area. The HC, on the other hand, climbed (or drifted) its way into this local optima, and was unable to escape thereafter. For the *FireVariance* task, only 9 searches ended up in the inferior local optimum, all of them being HC. That the local optima trapped more in the *FireDeriv* case is a little surprising, since the height of this local optima seemed more pronounced in the *FireVariance* landscape. However, the *FireDeriv* landscape has a

broader flat plateau between density 0 and 40, whereas the global optimum's basin extends out farther in *FireVariance*, providing a fitness gradient that helps lead more hill climbers that direction. Apart from these exceptions, all the searches eventually ended up on or around the global optimum / phase transition.

9.3.2.2. Segregation. In retrospect, the Segregation task may have been too easy a task for this benchmark collection. The performance curves over time (Figure 9.10) show the search dynamics, and all search methods quickly reach the optimum (or very close to it). Although the search methods were allowed 20K model runs, in all cases search performance had plateaued by 2000 model runs, and often much earlier. By the end of the search, all search methods have reached optimal levels of fitness (except for a few of the Sampling=1 cases which are very close, though not quite as high). Nevertheless, some search methods (GA-Gen, GA-SS, and RS) reach the best fitness values more quickly than the others. This was part of the motivation for using average fitness over time to measure overall search performance, rather than solely looking at end-of-search fitness, which would be uninformative in cases like this.

The ease of finding optimal solutions is explained by examining the fitness landscape, which is shown in Figure 9.11. A large portion of the space ($67 \leq$ pct-similar-wanted ≤ 75) yields optimal (or near-optimal) fitness values. The horizontal banding here is *not* a sampling artifact - the search space was sampled at high resolution in both dimensions. Rather, these bands are representative of discontinuities in model behavior depending on the value of the pct-similar-wanted parameter. The pct-similar-wanted parameter determines the threshold that agents compare their fraction of similar neighboring agents to, when deciding whether to move. Since each agent can only have up to 8 neighbors, agents can only ever observe

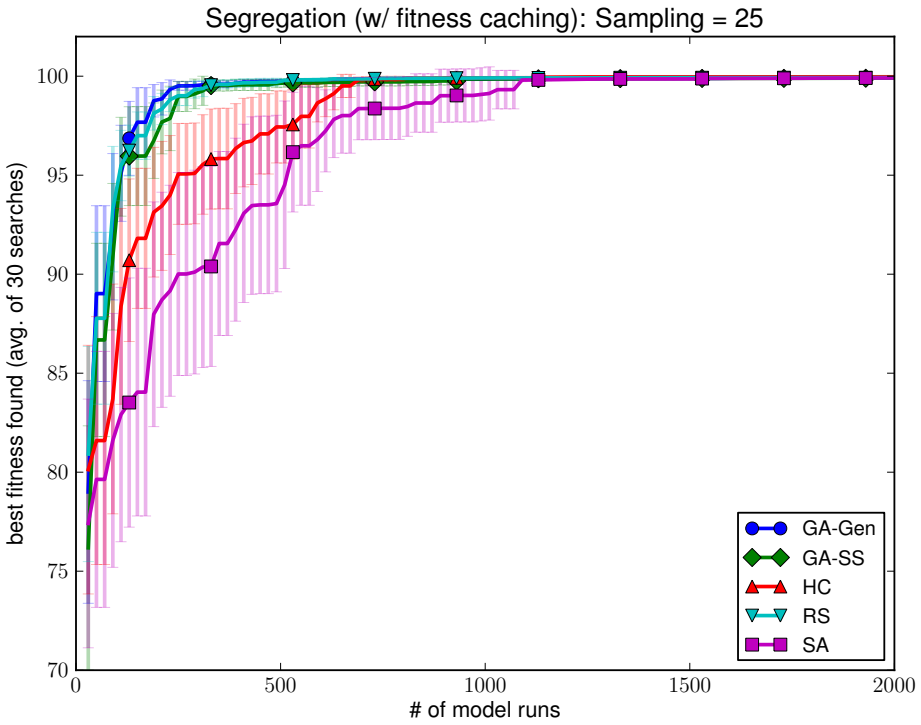


Figure 9.10. Search dynamics (performance over time) for each search algorithm on the Segregation task, with Sampling=25 and fitness caching turned on. (Error bars show 95% confidence intervals on the mean.)

fractions where the denominator is less than or equal to 8. Thus, changes to the pct-similar-wanted parameter only affect model behavior when the change crosses a possible fraction value. This creates large plateaus in the search space, which could potentially make it difficult for search algorithms, because they provide little search gradient for moving toward the high fitness region. However, the search space is fairly small, and the high fitness region is itself a large plateau, which results in making this an easy task. The crucial parameter in this fitness landscape is pct-similar-wanted, although in certain regimes (when pct-similar-wanted is either large or small) the number of agents also affects the outcome.

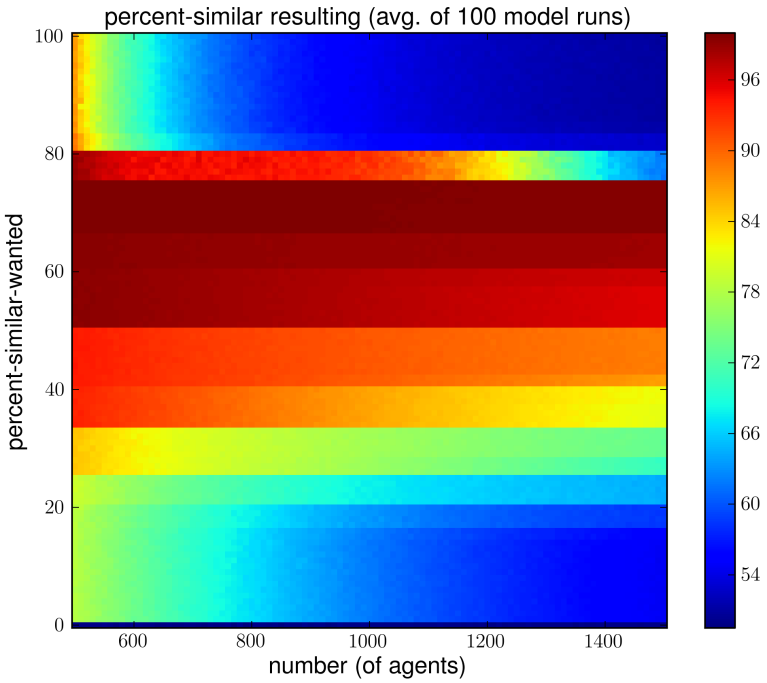


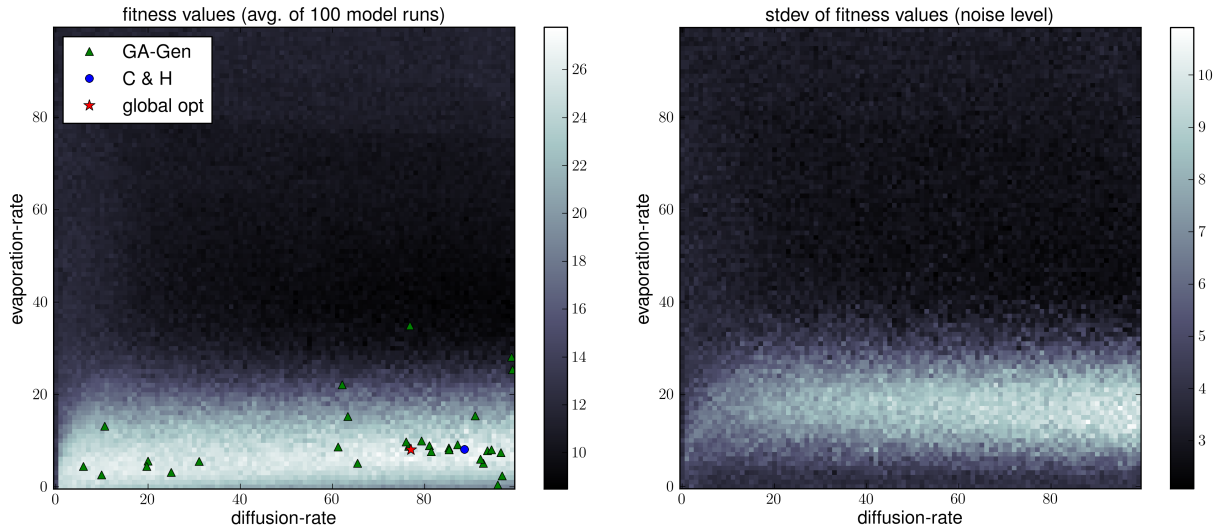
Figure 9.11. Fitness landscape for the *Segregation* task, calculated by exhaustive search of the space using 100 repeated model runs for each combination of parameters.

9.3.2.3. Ants. Although one reason for choosing the Ants task was to compare with Calvez and Hutzler's [2005] earlier experiment, a detailed comparison of search dynamics/performance and turned out to be infeasible, for two reasons:

(1) The best-fitness performance curves they show in their paper are biased by noisy sampling error, as they did not run independent trials to get an unbiased measure of fitness (as described above in Section 9.2.5).

(2) We could not implement their specific genetic algorithm and re-run it to compare, due to missing details in their paper necessary for replication of the experiment.

However, Calvez and Hutzler do report the results (parameter settings) discovered by their search, and we can compare these with the results discovered by our search methods given



(a) *fitness landscape* (b) *noise landscape*

Figure 9.12. Fitness (and noise) landscape for the Ants task. The best locations found by Calvez and Hutzler (C & H) [2005] and 30 GA-Gen searches (with caching and Sampling=25) are compared to the global best (from an exhaustive search). Substantial noise persists in the high fitness regions.

the same amount of computational effort (8120 model runs). These settings are diffusion-rate= 88.6, evaporation-rate= 8.1, and 100 independent trials (model runs) with these settings yield an average fitness value of 25.8 (with standard deviation of 7.6).

Unlike the previous two tasks, the exhaustive search of the Ants model was (necessarily) performed at a lower resolution than the search algorithms being benchmarked: the two parameters diffusion-rate and evaporation-rate were each varied from 0 to 99. The Ants fitness landscape, and associated noise landscape, are shown in Figure 9.12. From this exhaustive search, the best parameters found were diffusion-rate= 77, evaporation-rate= 8, yielding a fitness of 26.1 (with standard deviation 6.3) in 100 independent trials. Due to the high stochasticity of fitness evaluation, and it is not clear that this value is superior to that found by Calvez and Hutzler. In general, fitness values in this region are roughly

comparable - a large number of model runs would be necessary to determine statistical significance. Figure 9.12 also shows the best parameter settings found by 30 independent GA-Gen searches (with Sampling=25 and fitness caching on), after each search has run the model 8120 times. Although most of the searches have found good fitness regions, and some are quite close to the purported global optima, a few of the searches are still in poor fitness regions of the space. It is difficult to draw any conclusions, however, in comparison with Calvez and Hutzler's approach (an elitist GA with periodic temporal variation in sampling), because they only provide the results of a single search. In general, the search space is quite noisy, which can impede search, but there is a large region of high fitness, similar to the case with the Segregation task.

9.3.2.4. Fireflies. Although the 4-dimensional search space of the *Fireflies* task is small enough to enumerate, the higher dimensional nature of the space still makes it difficult to visualize. Several 2-D slices of the fitness landscape are shown in Figure 9.13. A key feature to note is that there is a nonlinear interaction between the number and flashes-to-reset parameters of the model. This results in a local fitness peak where number= 50 and flashes-to-reset= 2, from which a hill-climber could not climb directly to the more optimal region when number= 10 and flashes-to-reset= 1.

The best combination of parameters found in the exhaustive search was number= 10, flashes-to-reset= 1, cycle-length= 6, flash-length= 1, resulting in a fitness value of 0.80. These best settings for synchronization using the "advance" agent strategy still fall far short of the synchronization achievable when using the "delay" agent strategy, which can consistently achieve a synchrony value of 1.0. However, as we can see in Figure 9.13, there are many parameter settings that achieve much lower synchrony values than this (closer to 0.5). Only a relatively small portion of the search space has high fitness values, and there appear to

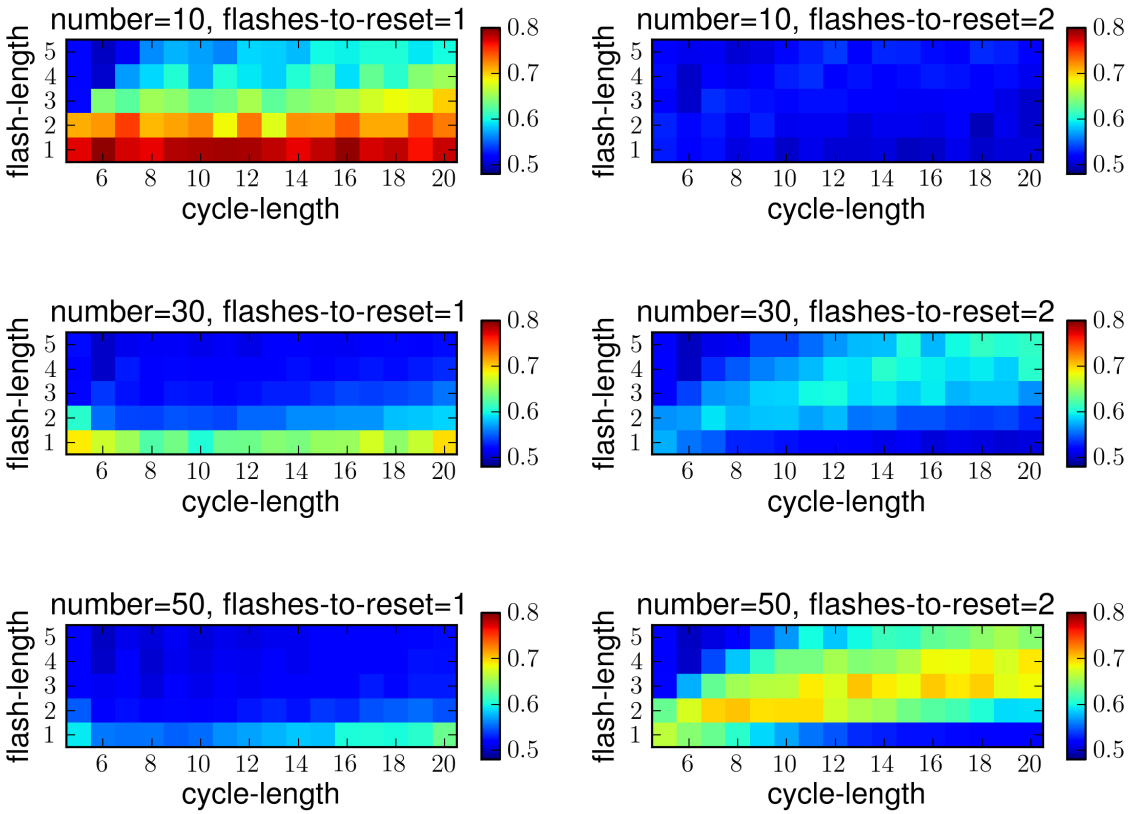
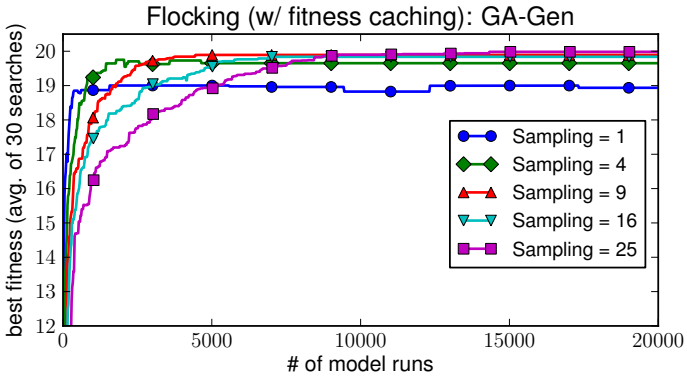


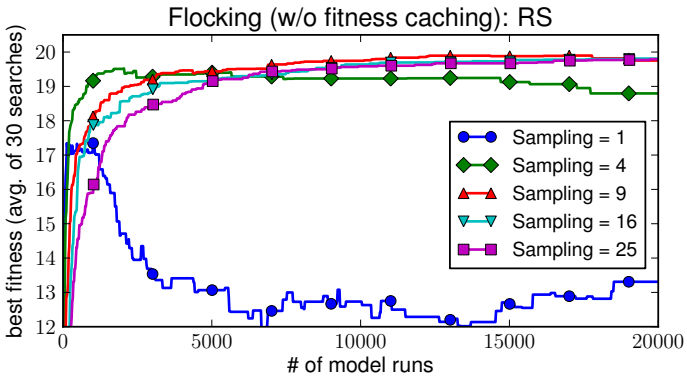
Figure 9.13. Selected slices of the fitness landscape for the *Fireflies* task, calculated by exhaustive search using 100 repeated model runs for each combination of parameters.

be inferior local optima elsewhere in the space (see, e.g., the medium-fitness region in the lower right panel of Figure 9.13), potentially making the fitness landscape more challenging to successfully navigate. However, the search algorithms were largely successful in finding high fitness values close to the global optimum; for example, 15 of the 30 GA-Gen searches (with caching off, Sampling=16) found solutions with fitness greater than 0.79, and all 30 had fitness greater than 0.76.

9.3.2.5. Flocking. With the *Flocking* task, we now move beyond the range of exhaustive search, and no longer have a full fitness landscape to compare with. Both genetic algorithms (GA-Gen and GA-SS) did well on this task, outperforming the other search methods using



(a)



(b)

Figure 9.14. Best-so-far performance for the *Flocking* task for GA-Gen with caching (a) and RS without caching (b), demonstrating the potential for negative impact of insufficient noise reduction.

the end-of-search performance measure in all cases except for Sampling=16 with caching (where it was bested by SA) and Sampling=9 without caching (where it was bested by HC). However, the most notable trend in the search dynamics for this task was related to noise reduction and sampling levels, and applied broadly to all search techniques. As shown in Figure 9.14, there was a definite pattern that when sampling was too low (Sampling=1, and to some extent Sampling=4), search performance was qualitatively worse than for higher sampling levels. This trend was not merely a matter of being slower to achieve good fitness values; when fitness evaluation was too noisy, the search performance plateaued or even

declined, suggesting that it would never find good solutions. As a result, when choosing a sampling level for noise reduction, it may be better to err on the side of too much sampling, rather than too little. With too much sampling, the search may progress more slowly, but practitioners will at least be able to see that the performance curves are still headed upward, and that if they run the search longer they are likely to find better solutions. With too little sampling, practitioners may be deceived into believing that the fitness plateau indicates that the search has reached the best value that can be found. Section 9.3.4 contains further discussion of sampling levels and noise reduction, examining trends more broadly across the benchmark tasks.

9.3.2.6. Daisyworld. The goal of the *DaisyWorld* task was to uncover a bug in the model, by finding a discontinuity in the search space. While fitness is measured on a continuum, being the amount of change between adjacent points in the space, what we are ultimately interested in is whether the searches arrived at parameter settings that would reveal the bug or not. However, as it turned out, apart from the model bug there were other locations in the search space where a small change in albedo-of-whites could result in a large change in the average number of daisies. Thus, some of the searches achieved high fitness but did not find the bug. Among the 300 GA-Gen searches, 219 reached fitness values greater than 15000, but only 88 of these were at a bug-noticing location. Among the search methods, the RS algorithm had the highest average performance overall for this task, but similarly it only found 83 bug-noticing locations, out of 237 searches that reached greater than 15000 fitness.

The fact that the goal (finding a bug) was not perfectly aligned with the fitness function (looking for large model changes that might represent either phase transitions or discontinuities) makes the results of this task difficult to interpret. However, in general, the task of finding a discontinuity somewhere in the search space resembles a "needle-in-a-haystack"

type of problem, and is unsurprising that more sophisticated measures (GA, HC, SA) do not outperform RS. At least, for this particular bug, there is no fitness gradient that can help lead searches to discover it – they essentially have to run across the hyperplane that the bug affects by chance.

9.3.2.7. Ethnocentrism. For the *Ethnocentrism* task, the GA-Gen algorithm performed very well, giving the best performance (using the average performance across time measure) on 9 out of 10 cases (5 noise sampling levels, with and without caching). It was only surpassed by GA-SS for the Sampling=16 without caching case. Under the end-of-search performance measure, GA-Gen also did quite well, although SA outperformed it in a few cases. GA-Gen (and GA-SS) performed substantially better than SA on the Sampling=1 case though, suggesting that they are better able to handle noisy fitness functions. A plot showing the effect of fitness caching on the search dynamics for the Ethnocentrism task appears later in Figure 9.16.

9.3.2.8. Wolf Sheep Predation. While the average best fitness achieved by any search method was only slightly greater than 1.0, some individual searches discovered parameters that yielded fitness as high as 1.4. Recall that the fitness function is the sum of the correlation for the wolves and the correlation for the sheep/moose. Thus, perfect correlation on both counts would result in a maximum fitness value of 2.0. This indicates that the search methods were moderately successful in finding parameters that could reliably generate good correlation with the real-world population trajectories. Given that the model was designed to be a simple abstract model of a predator-prey ecosystem, that there is considerable stochasticity in the model behavior, and that the real-world data is quite rough, a correlation of

around 0.7[14] for both populations is pretty good. It is also possible that a more sophisticated pattern-based matching measure (e.g., looking for periodic waveforms with similar frequency in the real and simulated data) might be more lenient in their calibration measure, and better highlight the similarities between the model and the Isle Royale ecosystem.

In terms of search performance, the GA-Gen algorithm had the best performance in more than half of the cases, but the highest average fitness out of all the cases was achieved by SA (shown in Figure 9.15). In this case the GA-Gen and GA-SS algorithms initially provide better performance early in the search, but are eventually overtaken by SA.

9.3.3. Efficacy of fitness caching

In addition to comparing search algorithms against each other, another goal of this benchmarking experiment was to empirically measure the benefit of fitness caching in the presence of noisy fitness evaluations. Table 9.7 shows whether caching had a positive or negative impact in each case, when measuring performance at the end of the search. Table 9.8 shows the same information, but for the performance measure that takes the average search performance over time.

Once again, these tables contain a large amount of information, which may be more easily digested in the summary form shown in Table 9.9. There are several results worth noting.

(1) Caching has little, if any, consistent effect on RS. For about half of the tasks the effect is positive, and in the other half it is negative. The negligible effect of caching here is logical, since the likelihood of RS randomly sampling the same point in the

[14]The actual breakdown for these parameter settings was 0.67 average correlation for the sheep/moose, and 0.74 average correlation for the wolves.

Model/Task	Sampling	RS	HC	SA	GA-Gen	GA-SS
FireDeriv	1	+	+	+	-	-
Segregation	1	-	+	+	+	-
Ants	1	+	-	-	-	-
Fireflies	1	+	-	-	-	-
Flocking	1	+	+	+	-	-
Daisyworld	1	-	-	+	-	-
Ethnocentrism	1	+	-	-	-	-
Heatbugs	1	+	+	+	-	-
WolfSheep	1	-	-	+	+	+
FireDeriv	4	+	+	+	+	+
FireVariance	4	+	-	-	-	-
Segregation	4	-	+	-	-	+
Ants	4	-	+	+	-	-
Fireflies	4	+	+	-	-	-
Flocking	4	-	-	-	+	-
Daisyworld	4	-	+	+	+	+
Ethnocentrism	4	-	+	+	-	-
Heatbugs	4	-	+	+	-	-
WolfSheep	4	-	+	+	+	+
FireDeriv	9	-	-	-	-	-
FireVariance	9	-	+	-	-	-
Segregation	9	+	-	+	+	+
Ants	9	-	+	+	-	-
Fireflies	9	-	+	+	-	+
Flocking	9	+	-	+	+	+
Daisyworld	9	+	+	+	+	-
Ethnocentrism	9	+	+	-	+	+
Heatbugs	9	-	+	+	-	+
WolfSheep	9	-	+	+	-	+
FireDeriv	16	-	+	+	+	+
FireVariance	16	-	+	+	-	-
Segregation	16	+	+	-	+	-
Ants	16	+	+	+	+	-
Fireflies	16	-	+	+	-	+
Flocking	16	-	-	+	-	+
Daisyworld	16	-	-	+	+	+
Ethnocentrism	16	-	+	+	+	+
Heatbugs	16	-	+	+	-	-
WolfSheep	16	-	+	+	+	+
FireDeriv	25	+	+	-	+	-
FireVariance	25	-	+	-	+	+
Segregation	25	+	-	+	-	-
Ants	25	-	+	-	+	+
Fireflies	25	+	+	+	+	+
Flocking	25	-	+	+	+	+
Daisyworld	25	+	-	+	+	+
Ethnocentrism	25	+	+	-	+	+
Heatbugs	25	-	+	+	+	+
WolfSheep	25	+	+	+	+	+

Table 9.7. Benefit of fitness caching when measuring performance at end of search. (+ indicates a positive effect, and - indicates a negative effect.)

Model/Task	Sampling	RS	HC	SA	GA-Gen	GA-SS
FireDeriv	1	+	+	+	-	-
Segregation	1	-	+	+	+	-
Ants	1	+	-	+	-	-
Fireflies	1	+	+	-	-	-
Flocking	1	-	+	+	-	-
Daisyworld	1	-	-	+	-	-
Ethnocentrism	1	+	-	-	-	-
Heatbugs	1	+	+	+	-	-
WolfSheep	1	-	+	+	+	+
FireDeriv	4	+	-	+	+	+
FireVariance	4	-	-	-	-	-
Segregation	4	-	+	+	-	+
Ants	4	-	+	+	-	-
Fireflies	4	-	+	-	-	-
Flocking	4	-	+	+	+	-
Daisyworld	4	+	+	+	+	+
Ethnocentrism	4	-	+	-	+	+
Heatbugs	4	-	+	-	-	-
WolfSheep	4	-	+	-	+	+
FireDeriv	9	-	+	+	-	+
FireVariance	9	+	-	+	-	-
Segregation	9	+	-	+	+	+
Ants	9	+	+	+	-	-
Fireflies	9	-	+	+	+	+
Flocking	9	-	+	-	+	+
Daisyworld	9	-	+	+	+	+
Ethnocentrism	9	+	+	+	+	+
Heatbugs	9	-	+	-	+	+
WolfSheep	9	+	+	+	-	+
FireDeriv	16	-	+	+	+	+
FireVariance	16	-	+	+	-	-
Segregation	16	+	+	+	+	+
Ants	16	+	+	+	+	-
Fireflies	16	-	+	+	-	+
Flocking	16	-	-	-	+	+
Daisyworld	16	+	-	+	+	+
Ethnocentrism	16	-	-	-	+	+
Heatbugs	16	+	+	+	+	-
WolfSheep	16	-	+	-	+	+
FireDeriv	25	+	+	-	+	+
FireVariance	25	-	-	+	+	+
Segregation	25	+	+	-	+	-
Ants	25	-	+	-	+	+
Fireflies	25	-	+	-	+	+
Flocking	25	-	+	+	+	+
Daisyworld	25	-	-	+	+	+
Ethnocentrism	25	+	+	-	+	+
Heatbugs	25	-	+	-	+	+
WolfSheep	25	-	+	+	+	+

Table 9.8. Benefit of fitness caching when measuring performance averaged across time. (+ indicates a positive effect, and - indicates a negative effect.)

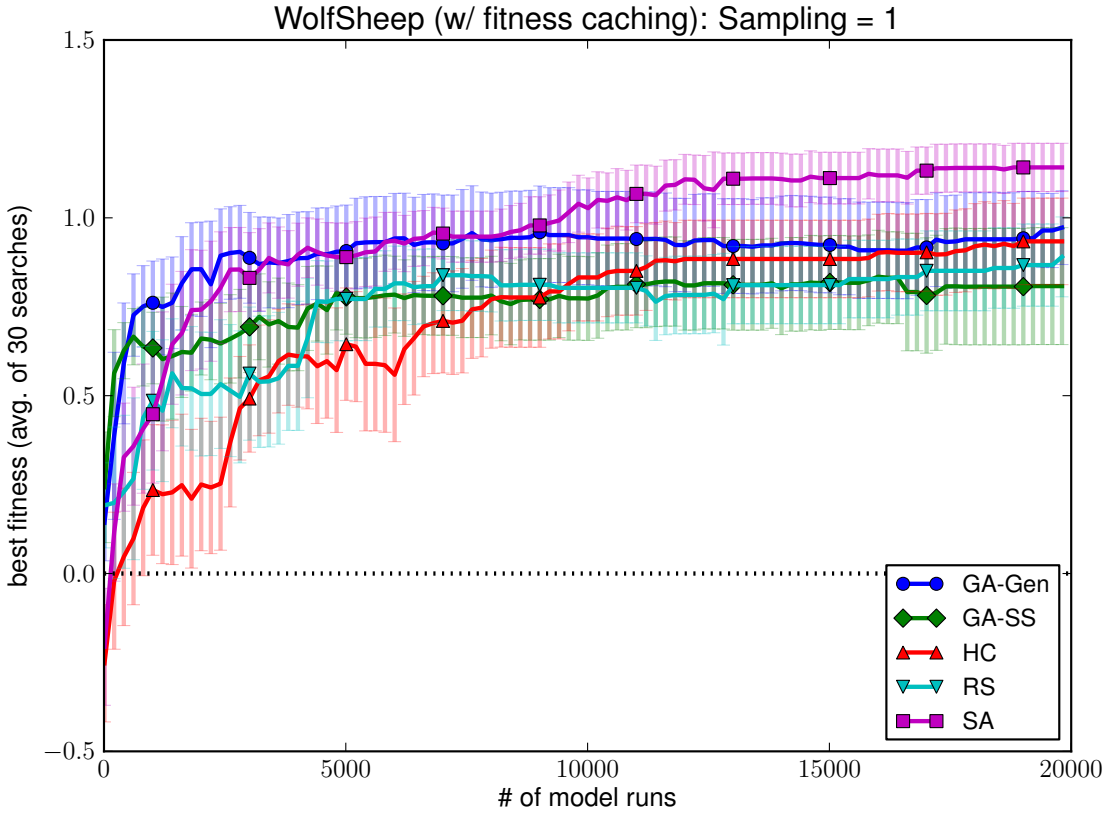


Figure 9.15. Best-so-far performance for each of the different search algorithm on the *WolfSheep* task with Sampling=10 and fitness caching turned on. Error bars show 95% confidence on the mean. Note that despite running each search 30 times, the confidence intervals are still fairly wide, meaning that search performance can vary substantially from one search to another.

search space twice is fairly low, and even if it does, it does not affect further choices about where to search in the space.

(2) Caching usually has a beneficial effect on both HC and SA, except for the lowest sampling levels (where the fitness landscape is noisiest), where caching is sometimes helpful and sometimes harmful. The result that caching is more helpful to HC and SA than to other methods likely stems from the fact that HC and SA operate more

	Search Algorithms					
Sampling	**RS**	**HC**	**SA**	**GA-Gen**	**GA-SS**	**Total**
1 sample	6/9	4/9	6/9	2/9	1/9	19/45
4 samples	3/10	8/10	6/10	4/10	4/10	25/50
9 samples	4/10	7/10	7/10	4/10	6/10	28/50
16 samples	2/10	8/10	9/10	6/10	6/10	31/50
25 samples	6/10	8/10	6/10	9/10	8/10	37/50
Total	21/49	35/49	34/49	25/49	25/49	140/245

(a) End-of-search performance measure

	Search Algorithms					
Sampling	**RS**	**HC**	**SA**	**GA-Gen**	**GA-SS**	**Total**
1 sample	5/9	6/9	7/9	2/9	1/9	21/45
4 samples	2/10	8/10	5/10	5/10	5/10	25/50
9 samples	5/10	8/10	8/10	6/10	8/10	35/50
16 samples	4/10	7/10	7/10	8/10	7/10	33/50
25 samples	3/10	8/10	4/10	10/10	9/10	34/50
Total	19/49	37/49	31/49	31/49	30/49	148/245

(b) Avg. across time performance measure

Table 9.9. Summary of the effects of caching, by search algorithm and noise sampling amount. Each cell shows the number of tasks where fitness caching was beneficial out of the number of possible tasks. (For *Sampling=1* the value is out of 9 rather than 10 because the FireVariance task was only run for higher sampling levels.)

locally than the other search methods. Thus are more likely to re-sample the same points in the search space repeatedly, giving a larger potential benefit for caching previously examined values.

(3) When fitness evaluation is noisy (low sampling levels), caching clearly has a harmful effect on GA performance. However, equally striking is the pattern that when the noise is reduced by sampling, fitness caching clearly improves GA performance. In particular, at the highest level of sampling (25 repeated model runs per fitness evaluation), fitness caching benefited GA-Gen in either 9 or 10 out of the 10 tasks

(depending on the performance measure). The trend is present for both GA-Gen and GA-SS.

The reason that the GA performance is more consistently impacted (both positive and negatively) by fitness caching at low and high noise levels is not entirely clear. With high noise, we hypothesize that the effects of frozen (cached) incorrect fitness values can be more detrimental to population-based search methods than other methods, since individuals with (falsely) apparent high-fitness may be difficult to weed out from the population, thus keeping search resources tied up in unproductive regions of the space. Also, population-based approaches have an implicit form of noise reduction by resampling those individuals who remain in the population over several generations, but this mechanism is thwarted when fitness caching is turned on.

Although most apparent for the GA search methods, there is a general pattern that fitness caching is more beneficial at higher sampling levels (reduced noise). This is not too surprising, considering that if fitness evaluation was completely deterministic (without noise), then fitness caching could not result in any loss of performance, and could only improve the situation. However, it is good to confirm this trend in the cases of intermediate noise, as our benchmarks show. It is also interesting to observe the extent to which the combination of high noise and fitness caching degrades GA performance.

To get more insight about how fitness caching affects search dynamics, we will zoom in to look at the search behavior over time, for a specific task: Ethnocentrism. The performance benefit due to fitness caching as the GA-SS search progresses is shown in Figure 9.16. In this figure, note the successive "peaks" in the curves plotting the caching benefit: first Sampling=1 peaks, then Sampling=4, Sampling=9, etc., followed by declines. As Figure 9.16 shows, fitness caching can provide a benefit to the search process early on in the search,

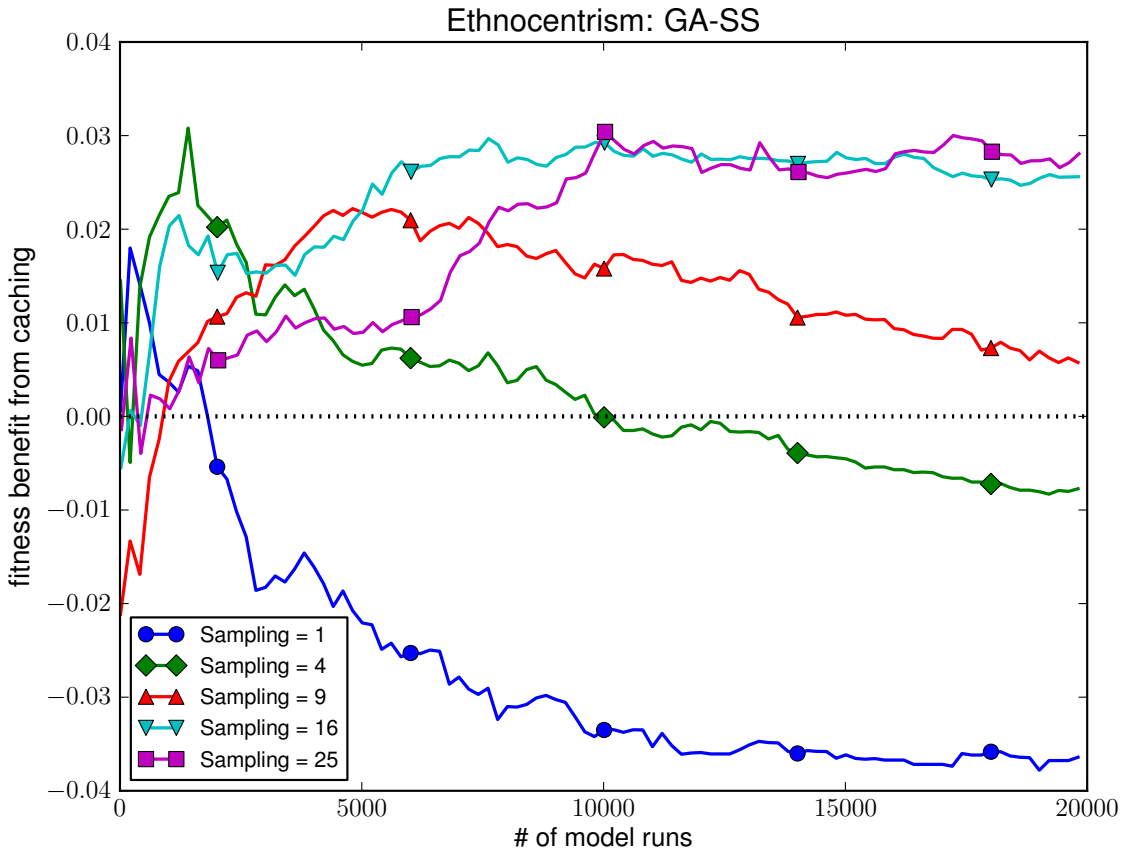


Figure 9.16. Caching benefit search dynamics in the Ethnocentrism task, for the GA-SS algorithm, with varying levels of sampling to reduce noise in the fitness evaluation. Benefit is measured as the difference between the average search performance (over 30 independent searches) with caching and without.

by allowing it to explore further more quickly (by avoiding recomputing previously tested solutions). However, when noise is substantial enough (i.e. Sampling=1 or Sampling=4), it eventually damages performance, possibly by preventing *fine-tuning* of solutions found by the GA – something that is possible when fitness caching is turned off. Although the effects of caching vary from one ABM exploration task to another, and sometimes results are quite noisy, the Ethnocentrism task illustrates a trend in GA performance (both GA-Gen and

GA-SS) that is also exhibited on several other tasks. For noisier fitness functions, caching is more likely to be beneficial early on in the search process, and detrimental later in the search. For less noisy fitness functions, it takes longer for the benefit of caching to be realized by the search, but the detrimental effect is avoided.

9.3.4. Noise reduction through varying levels of repeated fitness sampling

Before discussing the effectiveness of various levels of noise reduction, we first examine the amounts of noise present in the fitness landscapes. When search spaces are small enough, such as with the Ants model (Figure 9.12), it is possible to get a complete picture of the fitness noise throughout the whole search space. For larger spaces this is impossible, but we can still obtain a fitness noise profile by sampling a number of points in the space and measuring the variance (or standard deviation) of the behavioral measure at those locations. For uniformity (and convenience), we will exclude from this section's analysis those tasks that themselves involve measuring variance (*FireVariance*) those tasks that involve taking a derivative with respect to some parameter (*FireDeriv* and *DaisyWorld*), and those tasks that involve minimization rather than maximization (*HeatBugs*). For each of the remaining tasks, we chose 1000 points uniformly at random in the search space, and measured noise as the standard deviation of 10 replicate runs at those points. The distributions of noise throughout the space for selected tasks are shown in Figure 9.17, and the average (mean) noise level for each task is shown in Table 9.10.

For each task we also obtained an approximate ϵ-distribution (distribution of fitness differences between neighboring points in the search space, as defined previously in Chapter 8), and these are shown in Figure 9.18.

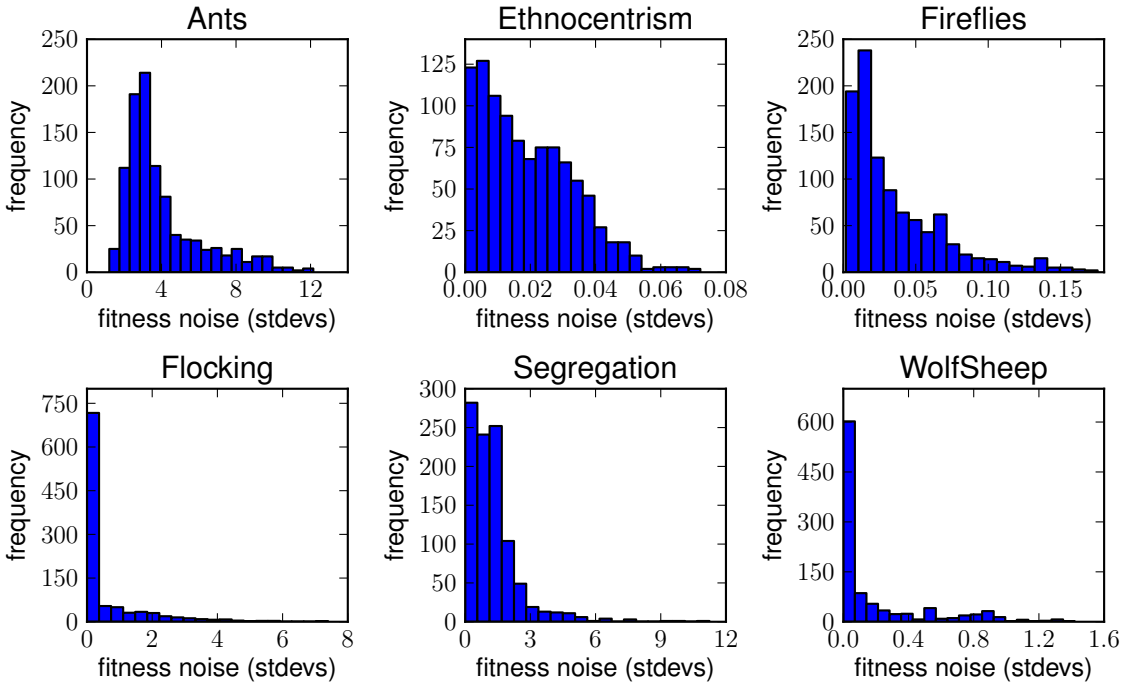


Figure 9.17. Distribution of noise in the search spaces for six of the ABM exploration tasks, as estimated from 1000 randomly sampled points in the search space, with 10 replicate runs at each point.

Model/Task	**Noise (stdev)**
Ants	4.01
Ethnocentrism	0.0191
Fireflies	0.0365
Flocking	0.539
Segregation	1.28
WolfSheep	0.192

Table 9.10. Average noise level for each task. Noise level is measured as the standard deviation of repeated behavioral measurements when running the model multiple times with the same parameter settings.

To further investigate the effects of noise reduction by repeated sampling in the presence of fitness caching, we use the probability of a *false switch* ($P(false\ switch)$) that we derived

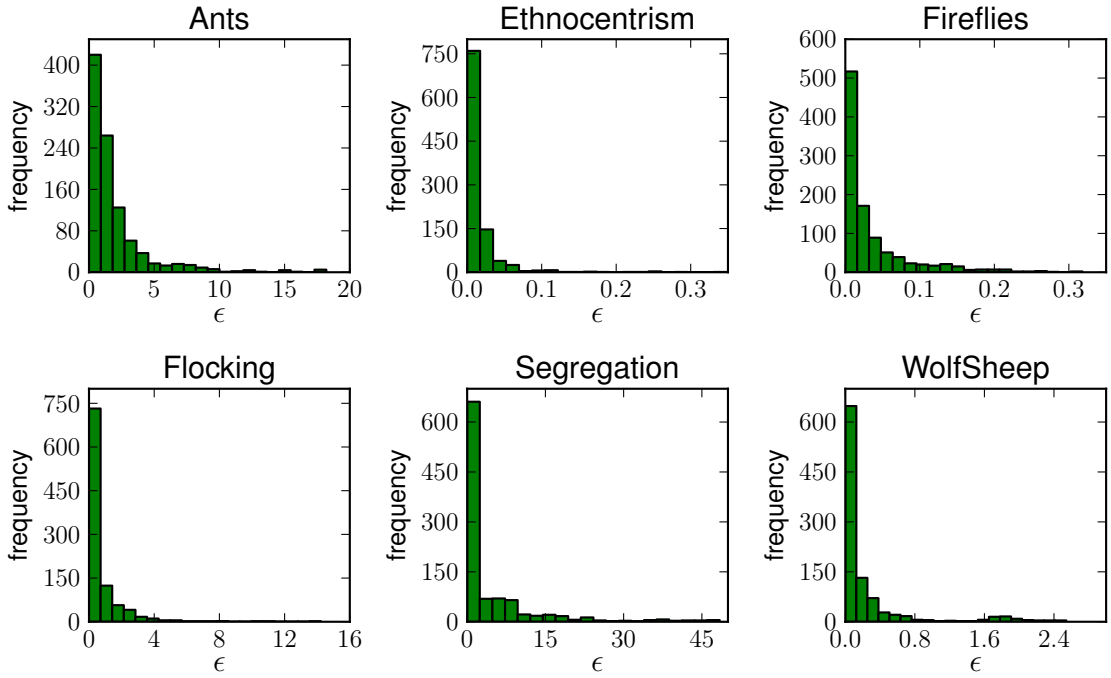


Figure 9.18. Distribution of differences between neighboring points in the search space (ϵ-distribution), for six of the ABM exploration tasks, as estimated from 1000 randomly sampled points in the search space, with 10 replicate model runs at each point and its neighbor (obtained by the mutation operator).

in Chapter 8[15]. Specifically, we calculate $P(false\ switch)$ by a numerical approximation of Equation 8.3, using the ϵ-distribution and mean noise value sampled from each task's fitness landscape. The resulting false switch probabilities for each task at each sampling level are shown in Figure 9.19. Notice the decreasing returns, in terms of the reduction of the likelihood of false switches occurring as you increase the amount of sampling to reduce noise.

[15]We do not use the *false optima* measure that was also derived in Chapter 8, because the benchmark search methods used Gaussian mutation, which does not permit calculation of the number of neighbors that each point has in the search space; rather, there is a probability distribution over possible neighbors.

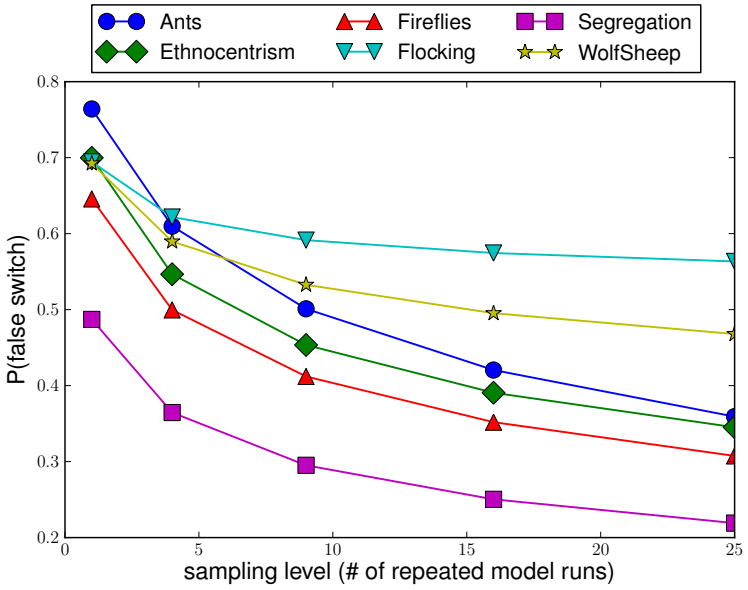


Figure 9.19. The probability of a *false switch* due to noise in the fitness evaluation, for each benchmark task and noise reduction sampling level.

We are further interested in the relationship between $P(false\ switch)$ and the performance of the various search methods. A plot of performance versus $P(false\ switch)$ for each task is shown in Figure 9.20. All other things being equal, a lower $P(false\ switch)$ is always better – but in this case, the lower $P(false\ switch)$ is only achieved by additional sampling. Because we fixed the total number of evaluations allowed for each search in these benchmarks, additional sampling means that the search will sample fewer points. Thus we find that in many cases, a medium level of $P(false\ switch)$ value often corresponds to the highest performance on a certain task. Unfortunately, different values of $P(false\ switch)$ achieve optimal performance on different models, which limits the usefulness of the $P(false\ switch)$ measure for predicting the optimal amount of sampling for noise reduction. This observation for these practical ABM benchmark tasks, along with the difficulties in predicting sampling levels for each of the artificial fitness landscapes in Chapter 8, suggest that the ϵ-distribution

for a given landscape is insufficient for precisely predicting the impact of noise on that landscape. Additional information about the fitness landscape, or the spatial distribution of noise within that landscape, appears to be needed in order to characterize the noise impact with reasonable precision. However, the best performance in these benchmark tasks for the genetic algorithms (GA-Gen and GA-SS) was always achieved for $P(false\ switch)$ values between 0.25 and 0.6. Thus, this information could serve as a rough (but useful) guideline for an upper bound on what $P(false\ switch)$ should be for effective genetic search; i.e., practitioners may wish to choose a sampling level sufficient to reduce the likelihood of a false switch occurring to below 60%. While a more precise value to predict optimal performance would be preferable, a guideline like this holds promise to help users avoid particularly bad search performance. Optimal noise reduction and sampling techniques for heuristic search are an active area of research, and much work remains to be done to improve methodology for choosing appropriate noise levels for efficient search.

9.3.5. Benchmark summary conclusions

Unfortunately for practitioners, the relationship between the performance of search algorithms and the noisy complex fitness landscapes created by ABM exploration tasks is neither simple nor straightforward. Complex interactions are involved, and different algorithms may be optimal for different fitness landscapes with certain features and search space dimensions. Despite this, there is some good news. In this chapter we have shown that genetic algorithms (specifically the generational GA) can perform well on a variety of search tasks with varying levels of noise. Although genetic algorithms may not be the optimal search method for some specific models and noise levels, in our benchmark tests it consistently provided good performance. Furthermore, the performance of the genetic algorithm appeared to be

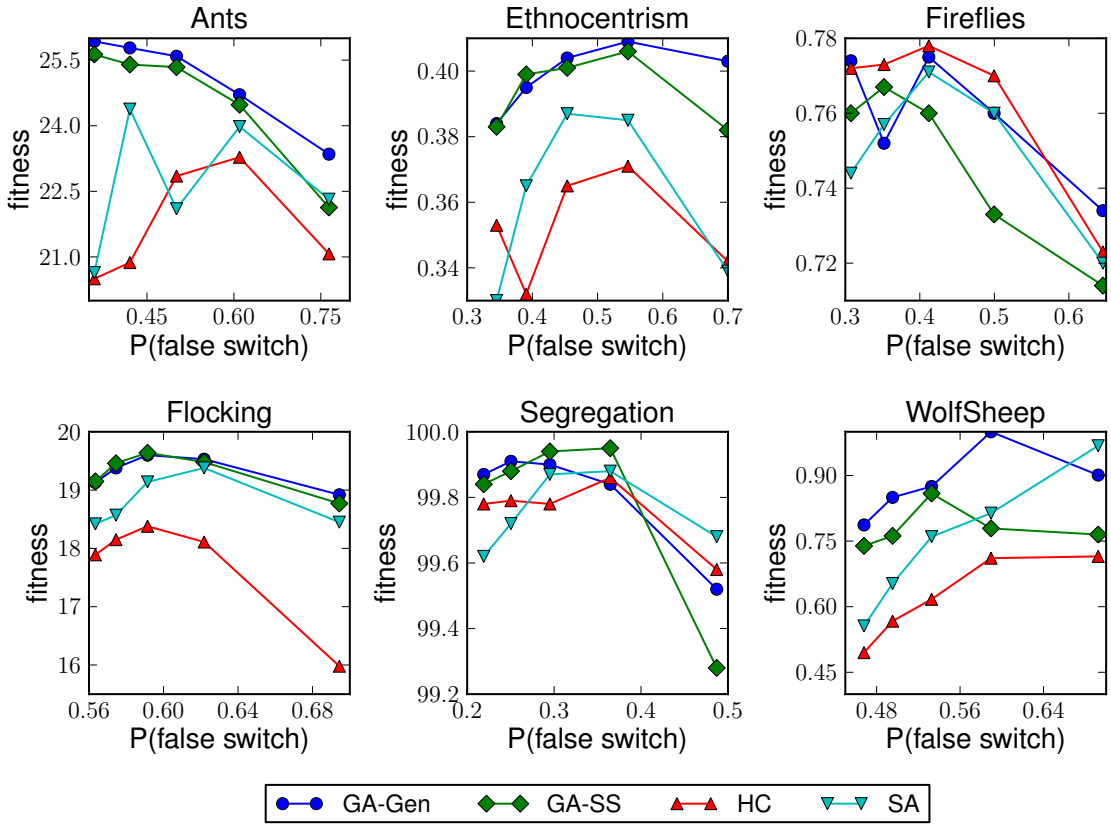


Figure 9.20. The probability of a *false switch* due to noise in the fitness evaluation, for each benchmark task and noise reduction sampling level (with fitness caching turned on).

improving, relative to the other search algorithms we examined, as the search spaces became larger and higher dimensional. This pattern is particularly encouraging, since many research-caliber ABMs have large numbers of parameters that need to be explored and analyzed. The benchmarks also demonstrated that the practice of fitness caching, even in the presence of noisy fitness evaluation, can be valuable for improving search performance, although not in the noisiest landscapes, unless sufficient noise reduction is performed. These benchmark tests, in conjunction with the ABM case studies presented in Chapters 4 through 7, show that genetic algorithms can rise to this challenge and provide an effective mechanism for

ABM exploration and analysis. As stated in the introduction to this chapter, although these benchmark experiments are far more comprehensive than any previous effort to characterize the performance of genetic algorithms (or other meta-heuristic search algorithms) in this domain, the results are not intended to be the final word on this subject. Instead, this chapter lays down the necessary groundwork for this type of research, and provides an open invitation for other researchers to follow in experimentation with alternative algorithms, methods, techniques and parameterizations, so that together we can provide practical guidance to the emerging area of QBME-style ABM analysis.

CHAPTER 10

BehaviorSearch: A New Tool for Metaheuristic ABM Parameter Search

"At each increase of knowledge, as well as on the contrivance of every new tool, human labour becomes abridged."

– Charles Babbage

"With four parameters I can fit an elephant, and with five I can make him wiggle his trunk."

– John von Neumann

The practice of designing and building new tools is crucial to computer science, and has been so since the early days of Charles Babbage's difference engine. Something which has changed is that most tools are now built in software, rather than at the hardware level. Compilers are an example of a software tool that fundamentally changed the landscape of computer science. However, many other tools have had substantial impact on the discipline, and society at large. The Google search engine is an excellent case in point. The theoretical ideas behind PageRank algorithm developed by Sergey and Brin [1998] were not entirely novel; in fact, similar graph-based ranking algorithms go back much further to ideas in information retrieval, bibliometrics, sociometry, and econometrics [Franceschet, 2011]. If Page and Brin had merely *talked* about the ideas of PageRank without acting on them to *build* a practical search engine (Google), the impact might have been limited to a few academic

journal papers and theoretical arguments. Instead, this tool revolutionized how people access and discover relevant content on the World Wide Web (incidentally, it also gave birth to a multi-billion-dollar company). There are, naturally, many other examples of the success of tool building, including the NetLogo [Wilensky, 1999] platform that *BehaviorSearch* interfaces with. Agent-based modeling lies at the intersection of computer science and many other disciplines, and as it is a growing field, there are many opportunities for building useful tools to serve this community. I am a strong proponent of the creation of tools to support new and innovative work in this domain, particularly in the analysis of ABM behavior, which is a challenging area with many AI applications.

With new tools comes new power, and with this power also comes responsibility. As is the case with many tools, *BehaviorSearch* [Stonedahl & Wilensky, 2010a] is one that may be alternatively used or abused, and thus I feel compelled to issue a warning, relating to von Neumann's glib remark about his ability to "fit an elephant". While von Neumann was referring to mathematical/statistical modeling methods, his broader point also applies to agent-based models. By giving your model enough parameters, your model can express a wide range of behavior. So if you want to show the world a model that displays elephant-trunk-wiggling behavior, *BehaviorSearch* can help you find parameter settings that will do that. Does the discovery of such parameters mean you have developed a good model? Not necessarily. It only means that the behavior you sought exists somewhere in the parameter space. Other questions must be considered: are the parameter assignments that caused this behavior reasonable? what effect does each parameter have on the outcome, and are those trends reasonable? One could use *BehaviorSearch* merely to calibrate/tune model parameters to highlight positive aspects of the model. However, it is the responsibility of the model author to carry out a critical analysis of the model – perhaps even to serve as

one of the model's severest critics. Fortunately, *BehaviorSearch* can assist with this process as well, to perform multi-variate sensitivity analysis (as with the Artificial Anasazi model described in Chapter 6), and to search for anomalous behavior that could be indicative of model errors.

In this chapter, we will discuss the design and implementation of *BehaviorSearch*, which is an open-source cross-platform tool that offers several search algorithms and search-space representations/encodings, and can be used to explore the parameter space of any ABM written in the NetLogo language. It was implemented in Java, and interfaces with the NetLogo modeling environment, using NetLogo's *Controlling API*. The user specifies the model file, the desired parameters and ranges to explore, the search objective function, the search method to be used, and the search space encoding, and then *BehaviorSearch* runs the search and returns the best results discovered (and optionally the data collected from all of the simulations run along the way). A beta-release of *BehaviorSearch* is freely available for download[1]. Our intent is to make advanced parameter search techniques accessible to a wide range of modelers so that the methods and ideas discussed in this thesis can be put into practice by others. We will begin by addressing the design goals, followed by a description of the software's current feature set (and how it supports those design goals). Finally, we discuss architectural and implementation details, including the ability to extend *BehaviorSearch* with new adaptive search algorithms and search space representations. This chapter will provide an overview of this software, but not detailed instructions for how to use it. For the latter, the reader is referred to the tutorial/documentation included with the software.

[1]Available at: `http://www.behaviorsearch.org/`

10.1. Design and Features of BehaviorSearch

10.1.1. Design Goals

BehaviorSearch follows in the tradition of NetLogo [Wilensky, 1999, 2001; Tisue & Wilensky, 2004], and Logo [Papert, 1980] before it, in embracing the twin design goals of "low threshold" and "high ceiling". By this we mean that the *BehaviorSearch* tool should be both easy for beginners to learn and use ("low threshold"), while also providing advanced features that will allow expert modelers to engage in cutting-edge research and analysis ("high ceiling"). To be clear, the "low threshold" goal for NetLogo, which aims to support use by elementary school students, is lower than that of *BehaviorSearch*, which primarily targets NetLogo's research audience. However, increasingly NetLogo is being used by undergraduates or even high school or middle school students who are developing agent-based models for research projects, and we would like *BehaviorSearch* to be accessible to these audiences, as well as researchers from various disciplines who are non-expert programmers but have adopted ABM methodologies for their research. Just as NetLogo strives to make the creation of agent-based models accessible to children and novices, BehaviorSearch aims to facilitate model analysis by making search and optimization techniques accessible to all modelers. The features that support these design goals are detailed in Sections 10.1.3 and 10.1.4 below. In conjunction with these design goals, *BehaviorSearch* is also designed to be extensible and open. This extensibility means that it should be possible to add new search algorithms and features in a simple modular way. Furthermore, the *BehaviorSearch* source code should be open and available to all, which provides several benefits that are discussed further in Section 10.2.

10.1.2. General features

Parameter-type flexibility. *BehaviorSearch* is capable of searching a combination of numerical (discrete/continuous), boolean, and categorical parameters. This is an important feature, since ABM parameters often take various forms, and are not constrained to always be of uniform type.

Search method variety. *BehaviorSearch* offers several different search algorithms and search space representations that users can employ. While the primary focus of my research has been on genetic algorithms, we have designed it as a general tool for applying any type of metaheuristic search algorithm to explore ABM parameter spaces. At present, *BehaviorSearch* supports the following search algorithms: random search, stochastic hill climbing, simulated annealing, and two variants of the genetic algorithm (*generational GA* and *steady-state GA*). We plan to add additional algorithms in the future. This flexibility is important since different approaches can be more or less effective for exploring different models.

QBME framework support. *BehaviorSearch* supports the QBME framework (described in Chapter 3) by providing options for different ways to condense data at different levels of ABM analysis: e.g., one can take the *median* value across model time steps, but then look at the *standard deviation* of results across multiple replicate runs with different random seeds.

Best-checking. As discussed in earlier in Section 9.2.5, *BehaviorSearch* provides built-in support of *best-checking*, to prevent users of the software from being misled by high fitness values resulting from ABM stochasticity (and so that users can easily detect if the search algorithm is being misled).

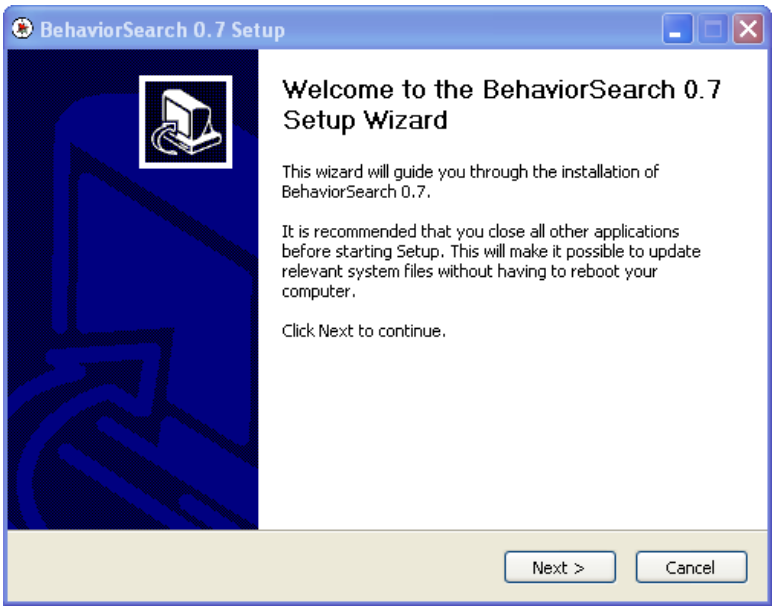


Figure 10.1. Windows graphical installer for *BehaviorSearch*.

10.1.3. "Low threshold" features

Graphical installer for Windows™. In an effort to make the software easy to use, the first step is making it easy to install. Since *BehaviorSearch* requires NetLogo to perform model runs, it needs to reside in a subfolder of the NetLogo installation folder. This can be a challenge, particularly on variants of the Windows operating system, where users may have difficulty finding the NetLogo installation folder, and may not have write-access privileges to modify its contents. As a result, we provide a graphical executable installer for Windows (see Figure 10.1) to simplify this installation process. (Installation on Mac/Linux computers is also reasonably straightforward, and generally just requires dropping a folder into the NetLogo application directory.)

Graphical User Interface. *BehaviorSearch*'s GUI provides an easy way to both design and run parameter searches. The dialog for designing experiments is shown in Figure 10.2,

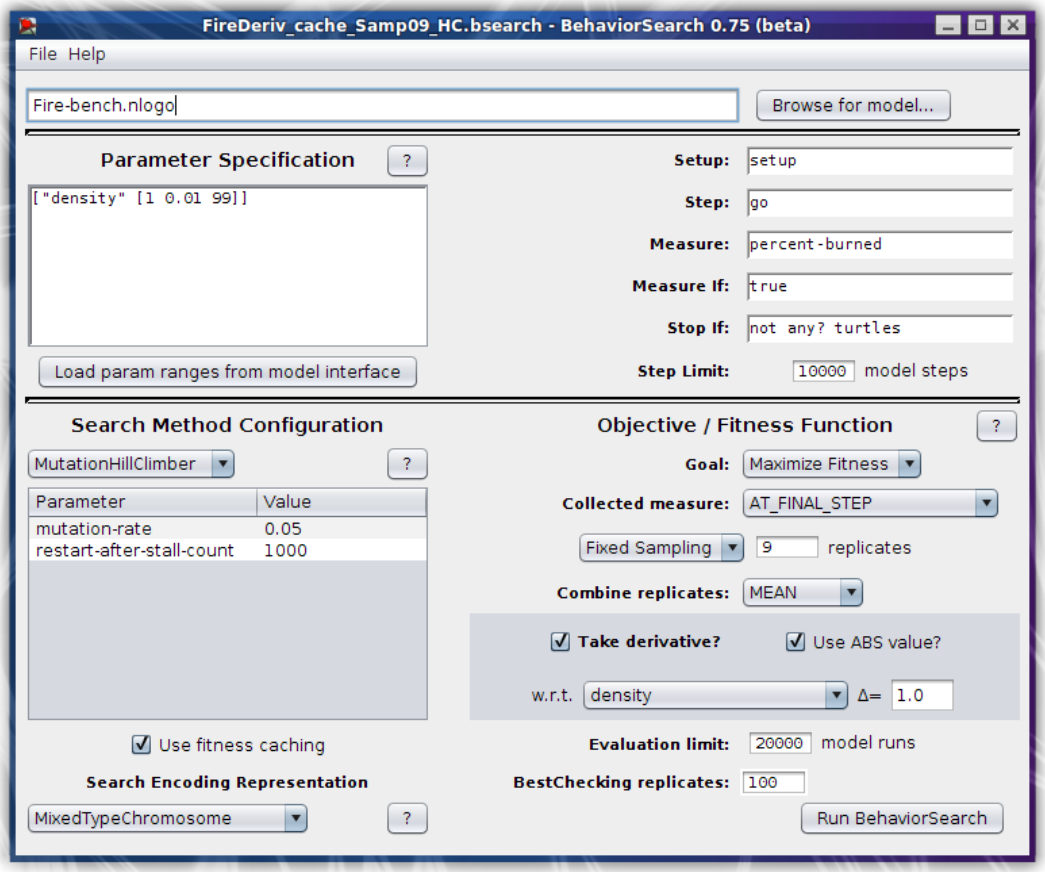


Figure 10.2. *BehaviorSearch* GUI for designing experiments involving model search, exploration, and optimization. This screenshot shows a search protocol for one of the *FireDeriv* exploration tasks discussed in Chapter 9.

and the dialog for launching parameter searches is shown in Figure 10.3. The GUI also provides error-checking and widget constraints to prevent users from creating malformed search queries.

Real-time feedback. *BehaviorSearch* provides integrated real-time search progress feedback. If searches are run from the *BehaviorSearch* GUI, then this information is displayed

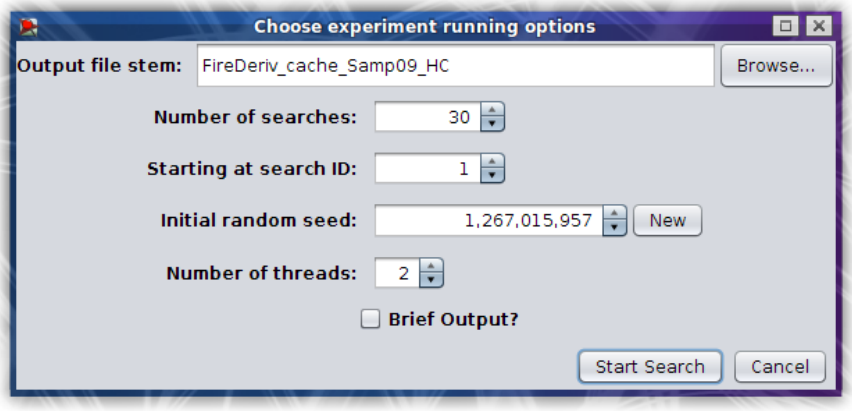


Figure 10.3. *BehaviorSearch* GUI dialog for launching search experiments.

via a progress bar and an auto-updating plot of fitness versus time (see Figure 10.4). Coupled with *best-checking*, this feedback permits the user to gauge search progress as it goes, as well as to get the latest information about the best model parameters found so far.

Natural learning progression. *BehaviorSearch* builds off of the success of NetLogo's built-in BehaviorSpace tool [Wilensky & Shargel, 2002], which provided a low-threshold way to perform grid/factorial parameter sweep experiments. Because the search space specification and model data collection were designed to be similar, users of NetLogo's BehaviorSpace tool should be able to get started quickly with *BehaviorSearch*. This design provides a natural learning progression: NetLogo (for building the model) ⇒ BehaviorSpace (for simple model analysis) ⇒ BehaviorSearch (for more advanced analysis and exploration).

Tutorial. *BehaviorSearch* ships with an included tutorial, which provides walk-through directions for using the tool to accomplish an example exploration task, as well as extensive coverage of the software's many features. All too often, academic software is thrust upon the world with insufficient information for users to even get started with it, let alone master its use. The great success and popularity of the NetLogo modeling toolkit is partially a

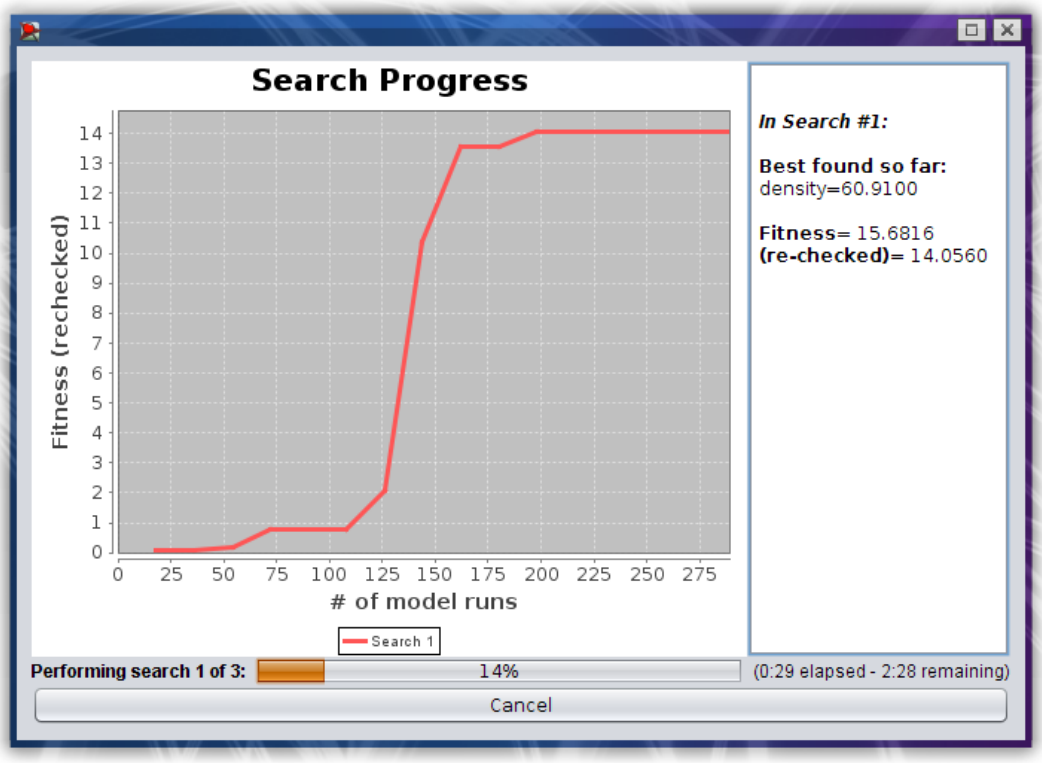


Figure 10.4. *BehaviorSearch* GUI dialog displaying search progress: this includes the fitness levels achieved as the search progresses, the best parameter settings found by the search so far, the percentage of the search that is completed, and an estimate of the amount of time remaining.

result of the thorough documentation and excellent tutorials that accompany it. We seek to replicate this success in *BehaviorSearch*, by ensuring that users have sufficient documentation to effectively learn how to use the software.

Integrated help and examples. Beyond the tutorial (which is available both on the *BehaviorSearch* website, and from the *Help* menu), the *BehaviorSearch* GUI provides integrated help/documentation via tooltips and localized help buttons which are only a click away. The non-GUI version of *BehaviorSearch* also provides help that documents each of the command line arguments/options for running. The software package also includes several example

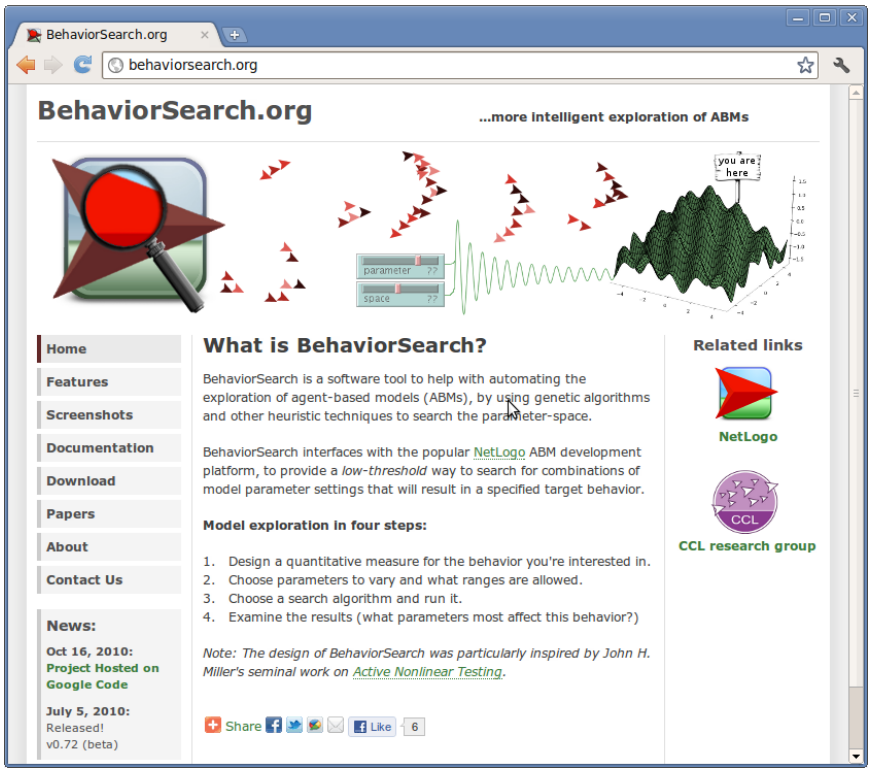


Figure 10.5. *BehaviorSearch* website, with supporting documentation and materials.

search experiment protocols, demonstrating how one might use *BehaviorSearch* to explore some of the models that come in NetLogo's models library.

Project Website. The *BehaviorSearch* tools is also supported by the accompanying project website (see Figure 10.5), located at `http://www.behaviorsearch.org/`. This website provides additional resources, such as a summary of features, information about new releases, links to relevant papers, and a contact form for user feedback. This site also links to a Google Code open source project website, with an issue/bug tracker, and access to the source code.

10.1.4. "High ceiling" features

Multi-resolution data output. *BehaviorSearch* can collect and store data at various levels of detail: recording each model run performed, each fitness evaluation, each time a new "best" is found, as well as the final best parameter settings at the end of each search. While novices can effectively use *BehaviorSearch* by simply looking at the final best parameters found, more advanced users can dig deeper into the search process and the results and parameters examined along the way.

Parametric derivatives. Built-in support for approximating derivatives of a behavioral objective function with respect to a specified parameter. As discussed in Chapter 3, this is useful for detecting phase transitions and critical points in the parameter space.

Parallelization and multi-threading support. *BehaviorSearch* was designed from the ground up with multi-threaded support[2] for parallel searching, offering improved performance for multi-processor/multi-core computers. As the number of cores in desktop computers proliferates, harnessing this parallelism becomes a crucial performance issue.

Command-line operation. *BehaviorSearch* includes a command-line version (see Figure 10.6) that is separate from the GUI version. This facilitates scheduling batch search operations, and more importantly, it allows one to run *BehaviorSearch* on remote clusters that do not have GUI support. There has been a substantial increase in recent years in both high performance computing clusters and cloud computing. By providing a command line version of *BehaviorSearch*, it is possible for users to spawn any number of independent searches and run them in parallel.

[2]Implementation note: this takes advantage of the improved concurrency features of Java 5/6, such as java.util.concurrent.ExecutorService and javax.swing.SwingWorker.

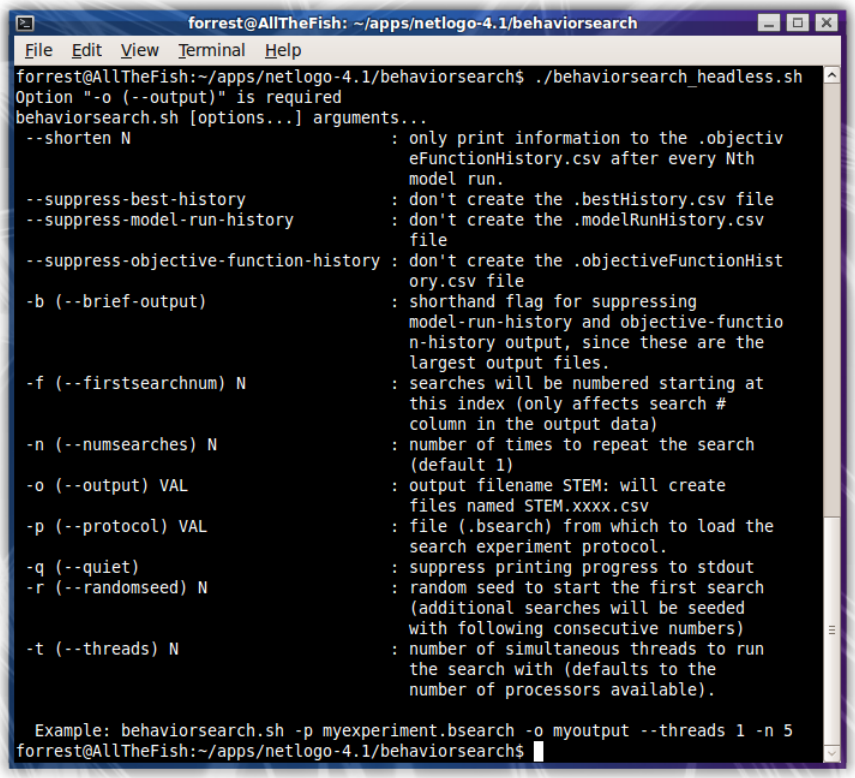


Figure 10.6. *BehaviorSearch* command line version. When *BehaviorSearch* is run without arguments, it displays all of the command line options/usage information, as shown above.

Extensibility. *BehaviorSearch* was developed using an extensible object-oriented framework, allowing new search algorithms and search space representations to be easily added, as will be discussed further in Section 10.2.

10.2. Architecture and Implementation

10.2.1. Open Source Status

BehaviorSearch has been released under an open-source license (specifically, the BSD 3-clause license[3]), which provides several important benefits.

[3]See: `http://www.behaviorsearch.org/LICENSE.TXT`

(1) This openness of source code provides transparency in academic research, so that researchers can fully examine the specific algorithms that are being employed to search the parameter spaces of their models. Failure to provide this level of transparency could compromise the legitimacy of this tool for serious research.

(2) This openness allows advanced users to customize the software to their own needs if necessary. The open source status guarantees complete extensibility.

(3) This openness further encourages community-contributed improvements to the software; users who are also programmers may submit bugfixes, add new features, and generally contribute to further enhancements of this project. The open source status is an invitation to become a member of a team effort in producing a tool that is useful to the community.

(4) This openness also promotes an open exchange of ideas, and may allow other ABM methodology researchers to design new tools to support query-based model exploration and analysis. Our goal here is to promote this research area (and practice in the field), not merely to promote our own specific tool for accomplishing it. The source code may assist others in developing alternative or derivative software that will push the field forward.

Because *BehaviorSearch* is open source software, the entire codebase is available for users to review, adapt, and extend, in case there are additional features which they require for their work. This provides the final rung in the ladder toward a "high ceiling"; expert users may dive under the hood to modify *BehaviorSearch* to meet their own needs if necessary.

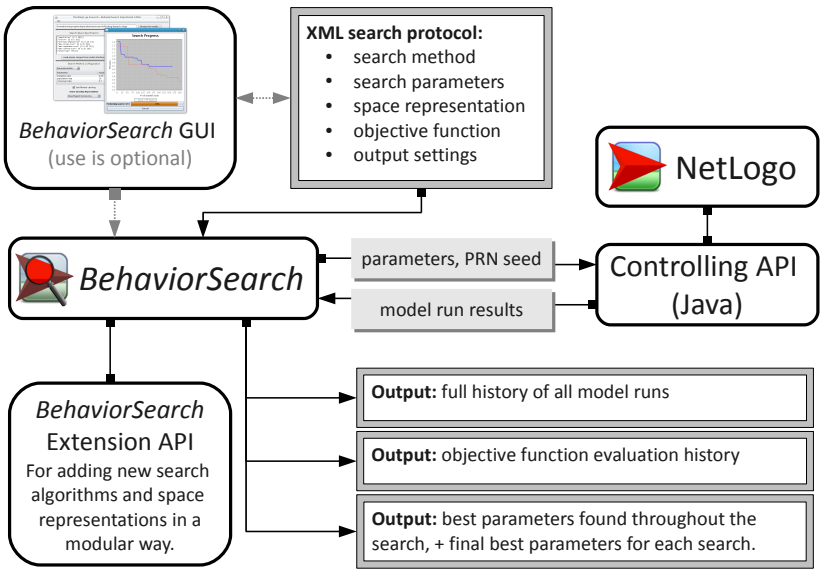


Figure 10.7. Design schematic of the *BehaviorSearch* architecture.

10.2.2. Modular Architecture

However, providing software with an open source license is no substitute for solid software design and clean architecture. Making public a messy spiderweb of code does little to promote extensibility or encourage community contributions. Thus, a modular design is crucial so that changes can be made to one part of the code without affecting others. Figure 10.7 shows a schematic of the overall architectural design, and also how *BehaviorSearch* interfaces with NetLogo. In particular, note that the *BehaviorSearch* engine is separated from the GUI layer, and does not depend on it.

At a finer level of detail, the *BehaviorSearch* codebase (written in Java) is divided into eight packages for organizational purposes:

- **bsearch.algorithms** - contains all of the search algorithms

- **bsearch.app** - contains the main code driving the *BehaviorSearch* application
- **bsearch.evaluation** - contains code for handling fitness evaluation and fitness caching
- **bsearch.nlogolink** - handles all of the communication with the NetLogo platform
- **bsearch.representations** - contains all of the search space representations
- **bsearch.space** - contains a representation of the parameter space
- **bsearch.test** - a package that contains unit testing
- **bsearch.util** - a package containing miscellaneous utility functions

As shown in Figure 10.7, *BehaviorSearch* also contains an extensions API. This API provides a clean interface for extending its capabilities via new search algorithms and search space representations, which will help support both continued research and any special needs of end users of the tool. To add a new search algorithm or search space, it is possible to simply write a new subclass of the AbstractSearchMethod class, drop it into the bsearch.algorithms package, and add one line to a text file that lists the search methods available to *BehaviorSearch*. The scenario for adding a new search space representation is similar. These plugin-style mechanisms are supported by the Java Reflection API. In general, the modular design of *BehaviorSearch* permits the addition of new functionality without editing any of the codebase, apart from the new Java class being added.

Search experiment protocols designed in *BehaviorSearch* are stored as XML documents (see Figure 10.8). Since XML is an industry-wide standard for data exchange, this provides compatibility for other tools to easily generate or manipulate search protocol files. The specific XML document format is formally specified by a DTD (Document Type Definition) which is included with the software package. The XML search protocol files also track

```
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE search SYSTEM "behaviorsearch.dtd">
<search>
<bsearchVersionNumber>0.73</bsearchVersionNumber>
<modelInfo>
<modelFile>Fire-bench.nlogo</modelFile>
<modelSetupCommands>setup</modelSetupCommands>
<modelStepCommands>go</modelStepCommands>
<modelStopCondition>not any? turtles</modelStopCondition>
<modelStepLimit>10000</modelStepLimit>
<modelMetricReporter>percent-burned</modelMetricReporter>
<modelMeasureIf>true</modelMeasureIf>
</modelInfo>
<fitnessInfo>
<fitnessMinimized>false</fitnessMinimized>
<fitnessCollecting>AT_FINAL_STEP</fitnessCollecting>
<fitnessSamplingReplications>9</fitnessSamplingReplications>
<fitnessCombineReplications>MEAN</fitnessCombineReplications>
<fitnessDerivative parameter="density" delta="1.0" useabs="true"/>
</fitnessInfo>
<searchSpace>
<paramSpec>["density" [1 0.01 99]]</paramSpec>
</searchSpace>
<searchMethod type="MutationHillClimber">
<searchMethodParameter name="mutation-rate" value="0.05"/>
<searchMethodParameter name="restart-after-stall-count" value="1000"/>
</searchMethod>
<chromosomeRepresentation type="MixedTypeChromosome"/>
<caching>true</caching>
<evaluationLimit>20000</evaluationLimit>
<bestCheckingNumReplications>100</bestCheckingNumReplications>
</search>
```

Figure 10.8. Example search protocol (in XML format) for one of the *FireDeriv* task experiments.

versioning, so that newer versions of *BehaviorSearch* that add new features to the protocol can seamlessly update search protocols created with prior versions.

10.3. Conclusion

Prior to the release of *BehaviorSearch*, QBME-style model analysis was only accessible to programmers who could either connect ABM toolkits to genetic algorithm (or other metaheuristic search) libraries, or write their own code from scratch. We have provided an easy-to-use tool with a graphical user interface that is integrated with a widespread and easy-to-use ABM platform (NetLogo), which constitutes considerable progress toward "low

threshold" ABM exploration. This tool also contains a variety of advanced features that we believe will be useful for serious ABM researchers, supporting a "high ceiling" so that this audience is not constrained. Furthermore, *BehaviorSearch* has been designed to be both open and extensible, to support customization and further development. In all these respects, *BehaviorSearch* provides a significant contribution to the *practice* of agent-based modeling. I believe this contribution is much needed, given the current state of the field: all too frequently ABMs are published and conclusions are drawn regarding these models, despite inadequate exploration of the model's parameter space, and insufficient model testing. We believe that a tool that is efficient, effective, and easy-to-use will lower the barriers to performing comprehensive ABM analysis and, given sufficient levels of adoption, will raise the community's standards for exploration of model behavior. Through informal conversations with researchers and modelers at conferences, workshops, and symposiums, I have confirmed that there is considerable interest in *BehaviorSearch*. It has also been discussed and recommended in ABM/complex systems workshops at the AAAI Fall Symposium in Arlington, Virginia and an NEH summer seminar on computational modeling in the humanities in Charlotte, N.C. Additionally, since the initial beta release, this software tool has been downloaded over 500 times. While this represents but a minute fraction of NetLogo's userbase (which numbers in the hundreds of thousands, comprised of both researchers and educational users), it still demonstrates significant interest in a small segment of early-adopters in the ABM researcher community. We anticipate a significant increase in adoption following the release of *BehaviorSearch* 1.0.

CHAPTER 11

Conclusions

"I've always been more interested in the future than in the past."
– Grace Hopper

"Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning."
– Winston Churchill

I share Grace Hopper's interest in the future, and have little desire to belabor the past. Accordingly, I will devote little time to the reiteration of points that were already made in the preceding ten chapters. Instead, I will merely provide a concise summary of the major contributions of this thesis, followed by a discussion of the broader impact of this work and applications to other research areas. The chapter concludes with a discussion of promising avenues for future work.

11.1. Contributions and Broader Impact

11.1.1. Contribution Summary

(1) We have provided the first comprehensive literature review (Chapter 2) of the intersection of research in genetic algorithms and ABM parameter analysis.

(2) We have extended Miller's [1998] work on Active Nonlinear Testing, and developed a new framework (Chapter 3) for using metaheuristic search methods (such as GAs) to explore and analyze ABM behavior.

(3) We have shown, through a series of case studies (Chapters 4 through 7), that genetic algorithms can be an effective method for exploring and discovering interesting behavior in serious modeling research in a variety of fields. These case studies also resulted in contributions to the application domains.

(4) We have derived new heuristics for quantifying the impact of uncertainty on noisy fitness landscapes (Chapter 8).

(5) We have designed a suite of ABM analysis benchmarks on a variety of models with varying levels of search space dimensionality and complexity (Chapter 9).

(6) We have and performed the first comprehensive experiment benchmarking performance of genetic algorithms against other comparable metaheuristic search algorithms (Chapter 9).

(7) We have designed and developed a new tool (*BehaviorSearch*), to allow ABM practitioners to easily apply genetic algorithms (and other metaheuristic search methods) to analyze the parameters of their models (Chapter 10).

11.1.2. Additional Applications and Implications

While in the preceding chapters we occasionally touched on the broader impact of the contributions of this thesis work, here we will provide some additional perspective on the connections to other applications or research areas.

Noise reduction methodology. In Chapter 8 and also Chapter 9 we investigated the impact of noisy fitness evaluation in conjunction with fitness caching. This line of research

is beneficial not only for ABM exploration, but also for other domains with slow noisy fitness evaluation. For instance, evolving cellular automata rules [Packard, 1988; Mitchell et al., 1996; Sondahl & Rand, 2007] is another area which is similar in this respect. For even moderate-sized lattices, it is impossible to evaluate the rules on all possible initial configurations, so fitness evaluations of the rules are only noisy estimates of the rule's true performance. Furthermore, fitness evaluation can be slow, depending again on the lattice size, and the number of iterations the CA rules are applied. Similar issues can arise in various industrial design problems – e.g., if the fitness of a design is evaluated by a lengthy discrete-event simulation that uses randomly generated scenarios to test a product's performance. Interactive GAs [Caldwell & Johnston, 1991] are another notable example where this analysis could be applicable, as they use humans to evaluate fitness, thus causing fitness evaluation to be typically both noisy and (very) slow.

Multi-agent systems and multi-agent learning. The field of agent-based modeling is closely related to, and sometimes overlapping with, the field of "multi-agent systems" (MAS). Agent-based modeling, which has been the primary concern of this document, attempts to simulate (or reproduce) the emergence of some natural or artificial phenomenon, through the interactions of multiple agents. In multi-agent systems, there is often a goal of engineering a multi-agent system to accomplish some specified task. For instance, the goal could be having a team of small autonomous fire-fighting robots collaboratively extinguish a burning building. In this case, the researcher's challenge may be to discover a set of rules by which these fire-fighting robot agents should act, such that they will exhibit a reliable strategy for extinguishing a burning building in a wide variety of real-world settings. Another example of multi-agent systems research would be the development of software agents that engage in independent bidding in online auctions/markets. A third example of multi-agent systems

research is the classic RoboCup competition [Kitano, Asada, Kuniyoshi, Noda, & Osawa, 1997], wherein teams of robots play soccer against one another. Multi-agent systems research is often coupled with the study of "multi-agent learning" - that is, techniques that agents (or teams of agents) can use to improve their performance on some goal task over time. For more information about the area of multi-agent learning, see the excellent survey by Panait and Luke [2005]. A primary difference between agent-based modeling and multi-agent systems research stems from the intended objective: is it to create a model for the sake of better understanding a system, or is it to create a system that performs some useful task? However, there is not always clear-cut agreement on the precise definition of either ABM or MAS (for instance, some consider MAS to be a sub-genre of ABM, and vice versa) and it is a blurry line that divides these two areas of research. The semantic debate aside, much of this thesis work will be useful for multi-agent system research as well as ABM research. For example, when designing a team of virtual soccer-playing bots, there will naturally be a number of global parameters associated with the bots (such as maximum-kicking-speed) and the environment (such as grass-friction-coefficient). It would be informative to explore the parameter-space of this multi-agent system, to calibrate the agent properties for optimal performance, or to discover the changes in environmental properties that the bots are most sensitive to. In some cases, multi-agent learning problems can be recast as parameter search problems, in which case, using a genetic algorithm to search for a specified behavior in the parameter space of a MAS is equivalent to evolving agents (or teams of agents) capable of performing a desirable task. However, sometimes the agent-level behavior is not best characterized by a set of global parameters: for instance, the team of agents may require heterogeneous behaviors, or agent behavior may be best represented as a flexible computer program, rather than fixed rules with parameters filling in the blanks, in which case a different technique such

as genetic programming (GP) may be more appropriate. However, the applicability of the theories, methods, and tools developed in this thesis spill over into multi-agent systems to some extent, and in some cases they even assume a new interpretation as solving multi-agent learning problems.

Evolutionary testing and verification in stochastic systems. As mentioned above, one of the thesis contributions was to develop a methodology and framework for using search-based techniques to address a variety of agent-based model exploration tasks, including some aspects of model testing and verification (such as sensitivity analysis leading to the discovery of a bug in the Artificial Anasazi model in Chapter 6).

There are numerous applications where there is a stochastic computational system (possibly hardware, software, or a network of computational devices) whose behavior is dependent on a number of parameters. In general, it is a difficult task to verify that the system is operating correctly, and that various types of anomalous behavior cannot occur. However, it may be helpful to use search techniques to evolve parameters that lead to various types of extreme behavior. If the extreme behavior is outside the range of acceptable behavior, there is evidence of a problem. Additionally, evolutionary search methods can test whether small perturbations in combinations of parameters can yield significantly different behavior. Thus, the work presented herein regarding the use of genetic algorithms for testing/verification of agent-based models may extend to broader software/system testing applications.

11.2. Future Work

As stated above, this dissertation is not intended as the "final word" on this subject, and it would be incomplete if it did not also identify key challenges that remain, as well

as provide guidance for future areas of fruitful research. This thesis has laid the necessary groundwork for several important research directions.

11.2.1. Further benchmarking

The development of a set of benchmark models and exploration tasks (Chapter 9 provides the necessary substrate for further experimentation regarding which metaheuristic search methods are most effective for agent-based model exploration. Although this thesis presented the most comprehensive comparison of search algorithms in this domain to date, much work remains to be done. Here are the three most promising lines for continuing research in this vein:

(1) Due to pragmatic constraints, the parameters of the search algorithms themselves were not varied. While sensible default values were chosen, additional experiments may reveal important trends in GA parameters such as the *mutation-rate*, *crossover-rate*, or *population-size*.

(2) There are several other metaheuristic search algorithms that should be tested – e.g., particle swarm optimization [Kennedy et al., 1995], harmony search[Geem et al., 2001], or the *cross-entropy* method [Rubinstein & Kroese, 2004].

(3) Although the models and tasks chosen for benchmarking represent a reasonably broad spectrum of QBME analysis, it is not comprehensive. Additional models and tasks may be desirable to include in benchmarking, and some of the current models and tasks should be reconsidered (for instance, the *Segregation* task may be too easy to be useful for evaluating search algorithm performance).

11.2.2. Multi-objective exploration

In a footnote in Chapter 3, we mentioned that it is not truly necessary to condense all of the information about a model run down to a single number, in order to search for behavior in the model. In fact, there is a thriving area of research called multi-objective optimization which focuses on searching for parameters that maximize (or minimize) multiple quantities simultaneously, and evolutionary algorithms have been shown to perform well on such tasks [Deb, Agrawal, Pratap, & Meyarivan, 2000; Zitzler, Laumanns, Thiele, et al., 2001; Deb, 2001]. There is also a body of research attempting to bridge work on multiobjective optimization with agent-based modeling and multi-agent systems [Socha & Kisiel-Dorohinicki, 2002; Rogers et al., 2004; Narzisi et al., 2006]. The extension of the QBME framework to handle multiple objective functions is reasonably straightforward. Instead of choosing a single behavior, practitioners would choose multiple behaviors, and design multiple objective functions that quantify those behaviors. For each search, the multi-objective search algorithm would return a set of Pareto-optimal parameter settings, rather than just a single "best" choice for the parameter settings, and users would be able to examine the trade-offs between the various behaviors, and the extent to which the behaviors could be elicited from the model simultaneously (using the same parameter settings). This is how it would work in theory. In practice, *BehaviorSearch* would need to be extended to include multi-objective search algorithms (such as SPEA2 [Zitzler et al., 2001] or NSGA-II [Deb et al., 2000]), and provide additional support for effectively visualizing the resulting Pareto-fronts (which poses challenges for high-dimensional datasets).

11.2.3. Additional Case Studies

Each of the case studies (Chapters 4 to 7) provided new insights or perspectives on exploration and analysis tasks in ABM research domains. Thus, additional case studies are also likely to be helpful and informative. I have been working on an ABM that I developed in conjunction with colleagues in the Linguistics Department at Northwestern University regarding how language can change or evolve in a social network context. Of particular interest is the conditions under which "language cascades" are (or are not) likely to occur. During the course of this work, I applied *BehaviorSearch* to find parameter settings that were particularly conducive to cascades. As a result, we identified an important region of the parameter space that we had not yet considered. I have also been in communication with a researcher who has created an agent-based model of hybrid/alternative-fuel vehicle adoption, and who is interested in applying QBME methodology to calibrate the model parameters against real-world data. More extensive exploration of the affordances of search-based exploration in these or other models could be enlightening.

11.2.4. Fitness landscape characterization

For several of the tasks with smaller search spaces, we examined the complete fitness landscape. With the tasks with larger search spaces, exhaustive experiments like this are impossible. However, it still may be possible to partially characterize these fitness landscapes. For instance, one might be able to estimate the number of local optima present in the space, and the size of the basins of attraction for each of these optima. The presence of noise confounds efforts to characterize landscapes in this manner, and the noise must be taken into account. Thus, this process is by no means trivial. However, it would be useful to develop a scheme

for fitness landscape profiles, to characterize ABM tasks and possibly group them into similar categories. If it were possible to efficiently create these fitness landscape profiles, then search methods could be tailored to ABM analysis tasks on an individual basis, with the potential to drastically improve search performance. On a related note, it would be useful to have a method for ranking the "complexity" of model exploration tasks, since the number of parameters (search space dimensionality), the resolution for varying parameters, and the amount of "noise" in the results provide only an incomplete picture of the complexity of the task. This ranking would help with the development of appropriate benchmarks, and in judging search algorithms relative performance on tasks of varying complexity.

11.2.5. Recombination and epistasis

One key hallmark of classic genetic algorithms (in contrast to other stochastic search mechanisms, such as hill climbing, simulated annealing, or particle swarm optimization) is the use of a crossover operator, which permits recombination of information during the search's progress toward a solution. That such recombination is beneficial is related to the *building block hypothesis* [D. E. Goldberg, 1989], which posits that genetic algorithms work by combining small contiguous pieces of solutions ("building blocks") to create better solutions. However, for many problem domains the benefits of crossover are uncertain. An important area of future research is to consider the extent to which crossover is beneficial in the domain of ABM exploration, and whether crossover can be improved in this context. In previous work [Stonedahl, Rand, & Wilensky, 2008a], I proposed a novel crossover mechanism called CrossNet, which uses a flexible network as the underlying chromosomal structure rather than the typical linear genome ordering. The level of performance benefit that can be gained from

this expanded chromosomal representation is unclear at present, but it does provide a mechanism for incorporating *a priori* domain knowledge into the search process, through the specification of epistatic interaction links between parameters in the chromosomal network. This touches on a rather general question in AI: in what ways, and to what extent, should human intuition be combined with black box search algorithms, so as to best take advantage of the strengths of both? To entirely eschew the use of human knowledge of a domain during a search process is arguably foolish and inefficient, but alternatively, implanting too much human bias may prevent the evolution of unexpected and surprisingly good solutions. An investigation of this idea could: 1) provide a real testbed for the CrossNet mechanism, 2) contribute to the understanding of the role of crossover in genetic algorithms, and 3) have broader implications regarding the inclusion of domain-specific knowledge within the general metaheuristic framework of genetic algorithms.

11.2.6. Algorithmic and cognitive biases

In a footnote in Section 2.1, I asserted that computational search methods such as genetic algorithms have different biases than human researchers when exploring the parameter space of an agent-based model. The intuition here is that researchers' expectations will be strongly flavored by their cognitive representation of the model, and analogies to the target phenomenon being modeled. For instance, in a model of epidemics, humans would expect an increase in a parameter named "immunization-fraction" to result in less disease spread, or in a model of ant foraging, humans would expect an increase in ant-movement-speed to result in a greater quantity of food being harvested. These intuitions will often be correct, and thus may lead humans to explore the space of model parameters more intelligently and efficiently than a computational search algorithm. However, there may be cases where these

intuitions are false (either because of emergence leading to surprising results, or because the human misunderstands the parameter's name and impact on the model rules, or possibly because there is a bug/error in the model. Since the metaheuristic search algorithm lacks any representation of model structure, it has no expectations based on parameter names or analogies to the target phenomenon; it will not be misled by parameter's names. However, search algorithms do have their own biases that affect the way that they explore the parameter space. For instance, most search algorithms make the assumption that if a point in the parameter space yields good results/behavior, a nearby point in the space is also likely to be a good choice. However, these arguments that the biases are different are based merely on my own intuition, rather than experimental evidence. Although I feel confident that the biases will turn out to be different, I am less sure of exactly *how* they will be different, or to *what extent*. In any case, this would be a very interesting cognitive science experiment, comparing the way that humans and genetic algorithms go about exploring a model's parameter space. Possibly the amount of information given to the human about the model and target phenomena could also be controlled (i.e., by substituting Greek letters for the more descriptive model parameter names) to see how this would affect results. Beyond the merely academic interest, these cognitive experiments might provide insight into how computer search algorithms could be adapted to mimic human behavior to achieve more effective results, or how search algorithms might be tailored to specifically complement the deficiencies and biases present in human exploration behavior.

11.2.7. Unsupervised exploration

I am convinced that unsupervised search and exploration of ABM behavior is one of the most intriguing areas for future work. One arguable drawback of the QBME framework,

is that it requires users to formulate specific quantitative measures of behaviors they are interested in. What if no such specification was necessary, and we could design an intelligent search algorithm that could derive its own measures for the given model, in order to discover interesting points or regions of the parameter space? As a specific case, we could imagine a phase-transition-detection algorithm, which would scour the search space for drastic changes in output patterns based on small changes in the model's parameters. A key challenge here is that models produce a monumental amount of information, and sifting through that information to create relevant condensed measures of ABM behavior will be challenging. To simplify the problem, one might ask the user to enter a large number of "variables of interest" for the model - or possibly the information could be automatically extracted from the plots and monitors in the ABM's interface. Even in this simpler case, there are challenges to finding patterns among a medium number of temporally-varying outputs. One possibility would be to use a hybrid of genetic algorithms with unsupervised clustering algorithms (to identify similar behavioral regimes amongst the output data). It may also be possible to cast this problem within an *active learning* framework, wherein the machine learning algorithm must decide (based on past history) what sampling point in the search space would be most informative for refining the algorithm's hypothesis about clusters in the search space. Recently, Bramson [2009; 2010] has been advocating for the use of Markov models as a formal representation of ABM behavior. Although unproven, this representation may provide a richer representation than simply using a collection of numeric time-series. It would be interesting to test this theoretical approach in practice by training a Markov model (e.g., using the Viterbi algorithm [Viterbi, 1967; Forney Jr, 1973]) with varying model parameters and then perform clustering on the resulting Markov models to identify regimes in the parameter space. There are, in fact, a large number of machine learning methods that

might be employed, including decision/regression trees, neural networks, and support vector regressions. However, further research and experimentation will be required both to devise appropriate methods for applying these algorithms and to assess the effectiveness of these various approaches. The possibilities are tantalizing – an artificial intelligence agent could independently analyze ABMs and provide human researchers with a set of observations and points of interest. Collaborative human-AI research is an active area in general, and this application offers the potential to eventually revolutionize how scientific model analysis is performed.

11.2.8. Emergence and the art of quantifying the qualitative

Emergence is one of the key ideas in the field of complexity science; myriad systems, composed of simple interacting entities, exhibit emergent properties. Sometimes these emergent properties are relatively straightforward to measure. For instance, in a complex system of fireflies synchronizing, a completely synchronized state can be detected by noting when all of the fireflies both start and stop flashing at the exact same times. A simple measurement of partial synchronization could be counting the maximum number of fireflies that are flashing at any given instant, but what if there are disparate groups of fireflies synchronized at the local level, but not all the groups are synchronized at the global level? Perhaps a more sophisticated measure might analyze the Fourier decomposition of the signal generated by the history of flashes over time. Furthermore, in many cases emergent behavior can be visually observed and identified by humans much more easily than it can be quantified, or identified by computer systems. Consider the NetLogo Flocking model [Wilensky, 1998]: what is it about the aggregate behavior that brings the model to life, creating the resemblance of real birds or schools of fish? It is not the convergence to a steady state (either in

location or alignment), but rather the dynamic equilibrium of birds flowing in and around each other in structured, yet partially chaotic, patterns. Attempting to quantify complex dynamic patterns such as this is a challenging task, and developing robust methods for doing this would contribute significantly to complex systems research. Chapter 3 presented some work in this area, using the organizing principles of levels and diversity to generate a rich array of behavioral measures. Through the development of quantitative measures of specific emergent properties in several example models and case studies (Chapters 4, 5, 6, 7, 9), this thesis provides useful material for continuing research in this direction. Future work may be able to draw on the measures of ABM behavior provided in this thesis to develop a more universal framework, or find behavioral measures that apply broadly to a variety of models. Complex systems researchers have not yet converged on a formal/quantitative definition of the concepts of either *emergence* or *complexity*. Nevertheless, the ability of intelligent computer systems to detect or discover emergence promises to be an important (and exciting) area in coming years.

11.3. Last words

My hope is that the seeds that were sown in this thesis will bear fruit in an abundance of research using and refining the query-based model exploration framework, and further probing how genetic algorithms can improve ABM analysis. Thus, although the words have been taken out of their original context, the quotation of Winston Churchill (that prefaces this chapter) seems remarkably apropos in this instance. This final thesis chapter is not the end. It is not even the beginning of the end. But it is, perhaps, the *end of the beginning*.

References

Abrahamson, D., Blikstein, P., & Wilensky, U. (2007). Classroom model, model classroom: Computer-supported methodology for investigating collaborative-learning pedagogy. *Proceedings of the Computer Supported Collaborative Learning (CSCL) Conference*, *8*(1), 46–55.

Althaus, S., & Tewksbury, D. (2000). Patterns of Internet and traditional news media use in a networked community. *Political Communication*, *17*(1), 21–45.

Amaral, L., & Ottino, J. (2004). Complex networks. *The European Physical Journal B - Condensed Matter and Complex Systems*, *38*, 147-162. Available from `http://dx.doi.org/10.1140/epjb/e2004-00110-5` (10.1140/epjb/e2004-00110-5)

An, G., & Wilensky, U. (2009). From artificial life to in silico medicine: NetLogo as a means of translational knowledge representation in biomedical research. In A. Adamatzky & M. Komosinski (Eds.), *Artificial life models in software* (2nd ed.). Berlin: Springer-Verlag.

Anderson, M., Srolovitz, D., Grest, G., & Sahni, P. (1984). Computer Simulation of Grain Growth. I.–Kinetics. *Acta Metall.*, *32*(5), 783–791.

Anderson, P. (1972). More is different. *Science*, *177*(4047), 393–396.

Ankenman, B., Nelson, B. L., & Staum, J. (2008). Stochastic kriging for simulation metamodeling. In *Proceedings of the 40th conference on winter simulation* (pp. 362–370).

Arthur, W. (1994). Inductive reasoning and bounded rationality. *The American economic review*, *84*(2), 406–411.

Arthur, W. (1998). *Increasing returns and path dependence in the economy*. University of Michigan Press.

Ashlock, D. (2006). *Evolutionary computation for modeling and optimization*. Springer-Verlag New York Inc.

Axelrod, R., & Hammond, R. A. (2003). The evolution of ethnocentric behavior. *Midwest Political Science Convention, Chicago, IL*.

Axtell, R. L., Epstein, J. M., Dean, J. S., Gumerman, G. J., Swedlund, A. C., Harburger, J., et al. (2002). Population growth and collapse in a multiagent model of the Kayenta Anasazi in Long House Valley. *Proceedings of the National Academy of Sciences*, *99*, 7275–7279.

Bäck, T., & Schwefel, H. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, *1*(1), 1–23.

Balaji, P., Srinivasan, D., & Tham, C. (2007). Uncertainties reducing techniques in evolutionary computation. In *IEEE Congress on Evolutionary Computation, 2007. (CEC 2007)* (pp. 556–563).

Ballester, P., & Carter, J. (2004a). An effective real-parameter genetic algorithm with parent centric normal crossover for multimodal optimisation. *Lecture Notes in Computer Science*, 901–913.

Ballester, P., & Carter, J. (2004b). An effective real-parameter genetic algorithm with parent centric normal crossover for multimodal optimisation. In *GECCO '04: Proceedings of the 6th annual conference on Genetic and Evolutionary Computation* (pp. 901–913).

Bankes, S. (2002). Agent-based modeling: A revolution? *PNAS*, *99*(10), 7199–7200.

Bankes, S., & Gillogly, J. (1994). Exploratory modeling: Search through spaces of computational experiments. *Proceedings of the Third Annual Conference on Evolutionary Programming*, 353–360.

Bankes, S. C. (2002). Tools and techniques for developing policies for complex and uncertain systems. *Proceedings of the National Academy of Sciences of the United States of America*, *99*(Suppl 3), 7263.

Barabasi, A. (2003). *Linked: How everything is connected to everything else and what it means.* Plume.

Barabasi, A., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, *286*(5439), 509.

Baruh, L. (2009). Social Media Marketing: Web X.0 of Opportunities. In T. Dumova & R. Fiordo (Eds.), *Handbook of research on social interaction technologies and collaboration software: Concepts and trends* (p. 33-45). Idea Group, Inc.

Bar-Yam, Y. (1997). *Dynamics of complex systems.* Cambridge, MA, USA: Perseus Books.

Bass, F. (1969). A new product growth model for consumer durables. *Management Science*, *15*(5), 215-227.

Bass, F. M. (2004, December). Comments on "A New Product Growth for Model Consumer Durables". *Management Science*, *50*, 1833–1840.

Batty, M. (2007). *Cities and complexity: Understanding cities with cellular automata, agent-based models, and fractals.* The MIT Press.

Berry, B. J. L. (2002, May). Adaptive agents, intelligence, and emergent human organization: Capturing complexity through agent-based modeling. *Proceedings of the National Academy of Science*, *99*, 7187-7188.

Bongard, J. C., & Lipson, H. (2005, Aug.). Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation*, *9*(4), 361-384.

Bousmalis, K., Hayes, G. M., & Pfaffmann, J. O. (2007). Improving evolutionary algorithms with scouting. In *Progress in artificial intelligence* (Vol. 4874, pp. 247–258). Springer.

Bramson, A. L. (2009). Formal measures of dynamical properties: Tipping points. In *2009 AAAI fall symposium series.*

Bramson, A. L. (2010). Formal measures of dynamical properties: Robustness and sustainability. In *2010 AAAI fall symposium series.*

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, *30*(1-7), 107 - 117. Available from `http://www.sciencedirect.com/science/article/B6TYT-3WRC342-2N/2/63e7d8fb6a64027a0c15e6ae3e402889` (Proceedings of the Seventh International World Wide Web Conference)

Brown, D., Page, S., Riolo, R., Zellner, M., & Rand, W. (2005). Path dependence and the validation of agent-based spatial models of land use. *International Journal of Geographical Information Science*, *19*(2), 153–174.

Brueckner, S. A., & Parunak, H. V. D. (2003). Resource-aware exploration of the emergent dynamics of simulated systems. In *AAMAS '03: Proceedings of the second international joint conference on autonomous agents and multiagent systems* (pp. 781–788). New York, NY, USA: ACM.

Bryson, J. J., Ando, Y., & Lehmann, H. (2007). Agent-based modelling as scientific method: a case study analysing primate social behaviour. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *362*(1485), 1685–1698.

Buck, J. (1988). Synchronous rhythmic flashing of fireflies. *The Quarterly Review of Biology*, *63*(3), 265–289.

Burt, R. (1995). *Structural holes: The social structure of competition.* Harvard University Press.

Caldwell, C., & Johnston, V. S. (1991). Tracking a criminal suspect through face-space with a genetic algorithm. In *Proceedings of the fourth international conference on genetic algorithms* (pp. 416–421).

Calvez, B., & Hutzler, G. (2005). Automatic tuning of agent-based models using genetic algorithms. In *MABS 2005: Proceedings of the 6th international workshop on multi-agent-based simulation.*

Cantu-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, *10*(2), 141–171.

Capelo Junior, E., Castro Silva, J. L. de, Briel, M. H. L. van den, & Villalobos, J. R. (2008). Aircraft boarding fine-tuning. In *Proceedings of the 14th international conference on industrial engineering and operations management (ICIEOM-2008).*

Caporale, G. M., Serguieva, A., & Wu, H. (2009). Financial contagion: evolutionary optimization of a multinational agent-based model. *Int. Journal of Intelligent Systems in Accounting and Finance Management*, *16*(1–2), 111–125.

Carley, K., Fridsma, D., Casman, E., Yahja, A., Altman, N., Chen, L., et al. (2006). BioWar: scalable agent-based model of bioattacks. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, *36*(2), 252–265.

Carlson, A., & Copeland, J. (1985). Flash communication in fireflies. *The Quarterly Review of Biology*, *60*(4), 415–436.

Caruana, R., & Schaffer, J. (1988). Representation and hidden bias: Gray vs. binary coding. In *Proceedings of the sixth international conference machine learning* (pp. 153–161).

Černỳ, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, *45*(1), 41–51.

Chandrasekaran, D., & Tellis, G. (2007). A critical review of marketing research on diffusion of new products. In *Review of marketing research* (pp. 39–80). ME Sharpe.

Chattoe, E., Saam, N., & Möhring, M. (1997). Sensitivity analysis in the social sciences: problems and prospects. In K. Troitzsch, N. Gilbert, & R. Suleiman (Eds.), *Tools and techniques for social science simulation* (p. 273). New York: Physica-Verlag.

Chevalier, J., & Mayzlin, D. (2006). The effect of word of mouth on sales: Online book reviews. *Journal of Marketing Research*, *43*(3), 345–354.

Chi, M. T. H. (2005). Commonsense conceptions of emergent processes: Why some misconceptions are robust. *The Journal of the Learning Sciences*, *14*(2), pp. 161-199. Available from `http://www.jstor.org/stable/25473477`

Cobb, H. (1990). An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. *NRL Memorandum Report*, *6760*, 523–529.

Collier, N., & Sallach, D. (2001). *Repast: Recursive Porous Agent Simulation Toolkit.* University of Chicago.

Conway, R., Johnson, B., & Maxwell, W. (1959). Some problems of digital systems simulation. *Management Science*, *6*(1), 92–110.

Cucker, F., & Smale, S. (2007). Emergent behavior in flocks. *IEEE Transactions on Automatic Control*, *52*(5), 852–862.

Dean, J. S., Gumerman, G. J., Epstein, J. M., Axtell, R. L., Swedlund, A. C., Parker, M. T., et al. (2000). Understanding Anasazi culture change through agent-based modeling. *Dynamics in human and primate societies: Agent-based modeling of social and spatial processes*, 179–205.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms* (Vol. 16). Wiley.

Deb, K., & Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. *Complex Systems*, *9*(2), 115–148.

Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel problem solving from nature (PPSN) VI* (pp. 849–858).

De Jong, K. (1980, Sept.). Adaptive system design: A genetic approach. *Systems, Man and Cybernetics, IEEE Transactions on*, *10*(9), 566-574.

De Jong, K. (1993). Genetic algorithms are NOT function optimizers. *Foundations of Genetic Algorithms*, *2*, 5–17.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor, MI, USA.

Dodds, P. S., & Watts, D. J. (2004, May). Universal behavior in a generalized model of contagion. *Phys. Rev. Lett.*, *92*(21), 218701.

Domingos, P., & Richardson, M. (2001). Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 57–66).

Droste, S., Jansen, T., & Wegener, I. (2002). Optimization with randomized search heuristics – the (A)NFL theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, *287*(1), 131–144.

Ducheyne, E., De Baets, B., & De Wulf, R. (2003). Is fitness inheritance useful for real-world applications? *Lecture Notes in Computer Science*, 31–42.

Dutta-Bergman, M. (2006). Community participation and Internet use after September 11: Complementarity in channel consumption. *Journal of Computer-Mediated Communication*, *11*(2), 469–484.

Eagle, N., Macy, M., & Claxton, R. (2010). Network diversity and economic development. *Science*, *328*(5981), 1029.

Edmonds, B., & Bryson, J. J. (2004). The insufficiency of formal design methods – the necessity of an experimental approach for the understanding and control of complex MAS. In *AAMAS '04: Proceedings of the third international joint conference on autonomous agents and multiagent systems* (pp. 938–945). Washington, DC, USA: IEEE.

Epstein, J. M. (1999). Agent-based computational models and generative social science. *Complexity*, *4*(5), 41–60.

Epstein, J. M. (2002). Modeling civil violence: An agent-based computational approach. *Proceedings of the National Academy of Sciences of the United States of America*, *99*(Suppl 3), 7243-7250. Available from `http://www.pnas.org/content/99/suppl.3/7243.short`

Epstein, J. M. (2008). Why model? *Journal of Artificial Societies and Social Simulation*, *11*(4), 12. Available from `http://jasss.soc.surrey.ac.uk/11/4/12.html`

Erdős, P., & Rényi, A. (1959). On random graphs. *Publicationes Mathematicae (Debrecen)*, *6*, 290–297.

Erdős, P., & Rényi, A. (1960). On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, *5*, 17–61.

Euler, L. (1736). Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, *8*, 128–140.

Fehler, M., Klügl, F., & Puppe, F. (2004). Techniques for analysis and calibration of multi-agent simulations. *ESAW*, 305–321.

Fehler, M., Klügl, F., & Puppe, F. (2006). Approaches for resolving the dilemma between model structure refinement and parameter calibration in agent-based simulations. In *AAMAS '06: Proceedings of the fifth international joint conference on autonomous agents and multiagent systems* (pp. 120–122). New York, NY, USA: ACM.

Ferrari, P., & Nagel, K. (2005). Robustness of efficient passenger boarding strategies for airplanes. *Transportation Research Record: Journal of the Transportation Research Board*, *1915*(-1), 44–54.

Fisher, R. A. (1971). *The Design of Experiments*. Hafner.

Fitzpatrick, J. M., & Grefenstette, J. J. (1988). Genetic algorithms in noisy environments. *Machine Learning*, *3*(2), 101–120.

Fogel, D., Chellapilla, K., & Angeline, P. (1999). Inductive reasoning and bounded rationality reconsidered. *IEEE Transactions on Evolutionary Computation*, *3*(2), 142–146.

Fogel, L. J. (1966). Artificial intelligence through simulated evolution. *New York: John Wiley & Sons, 1966.*.

Forney Jr, G. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, *61*(3), 268–278.

Forrest, S., & Mitchell, M. (1993). What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. *Machine Learning*, *13*(2), 285–319.

Forrester, J., & Collins, J. (1969). *Urban dynamics*. Mit Press Cambridge, Mass.

Fox, R. (1971). *Optimization methods for engineering design*. Addison-Wesley Pub. Co.

Franceschet, M. (2011, June). Pagerank: standing on the shoulders of giants. *Communications of the ACM*, *54*, 92–101. Available from `http://doi.acm.org/10.1145/1953122.1953146`

Gantovnik, V., Anderson-Cook, C., Gürdal, Z., & Watson, L. (2003). A genetic algorithm with memory for mixed discrete–continuous design optimization. *Computers and Structures*, *81*(20), 2003–2009.

Geem, Z., Kim, J., & Loganathan, G. (2001). A new heuristic optimization algorithm: harmony search. *Simulation*, *76*(2), 60.

Gell-Mann, M. (1995). *The quark and the jaguar: adventures in the simple and the complex*. New York, NY, USA: W. H. Freeman & Co.

Gilbert, G. (2008). *Agent-based models*. Sage Publications, Inc.

Gilbert, N., & Bankes, S. (2002). Platforms and methods for agent-based modeling. *Proceedings of the National Academy of Sciences of the United States of America*, *99*(Suppl 3), 7197-7198. Available from `http://www.pnas.org/content/99/suppl.3/7197.short`

Gilbert, N., & Troitzsch, K. (2005). *Simulation for the social scientist*. Open University Press.

Gini, C. (1912). Variabilitá e mutabilitá. *Studi Economico-Giurdici Fac. Giursprudenze Univ. Cagliari, A*, *III*(Parte II), 3-159.

Glover, F., Kelly, J. P., & Laguna, M. (1996). New advances and applications of combining simulation and optimization. In *Wsc '96: Proceedings of the 28th conference on winter simulation* (pp. 144–152). Washington, DC, USA: IEEE Computer Society.

Goldberg, D. (1987). Simple genetic algorithms and the minimal, deceptive problem. In L. Davis (Ed.), *Genetic algorithms and simulated annealing* (pp. 74–88). London: Pitman.

Goldberg, D., Korb, B., Deb, K., et al. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, *3*(5), 493–530.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc.

Goldberg, D. E. (1991). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems*, *5*, 139–167.

Goldenberg, J., Han, S., Lehmann, D., & Hong, J. (2009). The role of hubs in the adoption process. *Journal of Marketing*, *73*(2), 1–13.

Goldenberg, J., Libai, B., Moldovan, S., & Muller, E. (2007). The NPV of bad news. *International Journal of Research in Marketing*, *24*(3), 186–200.

Goldenberg, J., Libai, B., & Muller, E. (2001). Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, *12*(3), 211–223.

Goldstone, R., & Janssen, M. (2005). Computational models of collective behavior. *Trends in Cognitive Sciences*, *9*(9), 424–430.

Goldstone, R., & Wilensky, U. (2008). Promoting transfer through complex systems principles. *Journal of the learning sciences*, *15*, 35–43.

Granovetter, M. (1973). The strength of weak ties. *The American journal of sociology*, *78*(6), 1360–1380.

Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, *16*(1), 122–128.

Grefenstette, J. (1992a). Deception considered harmful. In L. D. Whitley (Ed.), *Foundations of genetic algorithms* (Vol. 2, pp. 75–91). San Mateo: Morgan Kaufmann.

Grefenstette, J. (1992b). Genetic algorithms for changing environments. *Parallel problem solving from nature*, *2*, 137–144.

Grimm, V., & Railsback, S. (2005). *Individual-based modeling and ecology*. Princeton University Press.

Guimera, R., Uzzi, B., Spiro, J., & Amaral, L. (2005). Team assembly mechanisms determine collaboration network structure and team performance. *Science*, *308*(5722), 697.

Gumerman, G., Swedlund, A., Dean, J., & Epstein, J. (2003). The evolution of social behavior in the prehistoric American Southwest. *Artificial life*, *9*(4), 435–444.

Hansen, N., Finck, S., Ros, R., & Auger, A. (2009a). *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions* (Research Report No. RR-6829). INRIA. Available from `http://hal.inria.fr/inria-00362633/PDF/RR-6829.pdf`

Hansen, N., Finck, S., Ros, R., & Auger, A. (2009b). *Real-Parameter Black-Box Optimization Benchmarking 2009: Noisy Functions Definitions* (Research Report No. RR-6869). INRIA. Available from `http://hal.inria.fr/inria-00369466/PDF/RR-6869.pdf`

Harrison, J., Lin, Z., Carroll, G., & Carley, K. (2007). Simulation modeling in organizational and management research. *The Academy of Management Review (AMR)*, *32*(4), 1229–1245.

Hasan, A. A., Dellarocas, C., Lucas, H. C., & Yim, D. (2010). *The impact of the internet and online news on newspapers and voter behavior* (Tech. Rep.). University of Maryland.

Heppenstall, A. J., Evans, A. J., & Birkin, M. H. (2007). Genetic algorithm optimisation of an agent-based model for simulating a retail market. *Environment and Planning B: Planning and Design*, *34*, 1051–1070.

Heppner, F., Convissar, J., Moonan Jr, D., & Anderson, J. (1985). Visual angle and formation flight in Canada Geese (Branta canadensis). *The Auk*, 195–198.

Holland, J. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Holland, J. (2000). Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary computation*, *8*(4), 373–391.

Holland, J. (2008). Biology's gift to a complex world. *The Scientist*, *22*(9), 356–43.

Holland, J. H. (1995). *Hidden order: how adaptation builds complexity.* Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc.

Holme, P., & Newman, M. (2006). Nonequilibrium phase transition in the coevolution of networks and opinions. *Physical Review E*, *74*(5), 56108.

Hong, L., & Page, S. (2004). Groups of diverse problem solvers can outperform groups of high-ability problem solvers. *Proceedings of the National Academy of Sciences of the United States of America*, *101*(46), 16385.

Hong, L. J., & Nelson, B. L. (2006). Discrete optimization via simulation using COMPASS. *Operations Research*, *54*(1), 115–129.

Horn, J., & Goldberg, D. (1994). Genetic algorithm difficulty and the modality of fitness landscapes. In L. D. Whitley (Ed.), *Foundations of genetic algorithms* (Vol. 3). San Mateo: Morgan Kaufmann.

Horne, G. E., & Meyer, T. E. (2004). Data farming: Discovering surprise. In *Proceedings of the 36th conference on winter simulation* (pp. 807–813).

Huang, Y., Xiang, X., Madey, G., & Cabaniss, S. (2005). Agent-based scientific simulation. *Computing in Science & Engineering*, 22–29.

Hughes, E. (2000, September). *Multi-objective probabilistic selection evolutionary algorithm* (Tech. Rep. No. DAPS/EJH/56/2000). RMCS, Shrivenham, UK: Department of Aerospace, Power, & Sensors, Cranfield University.

Ilachinski, A. (2000). Irreducible semi-autonomous adaptive combat (ISAAC): An artificial-life approach to land combat. *Military Operations Research*, *5*(3), 29–46.

Iványi, M., Bocsi, R., Gulyás, L., Kozma, V., & Legendi, R. (2007). The Multi-Agent Simulation Suite. In *Emergent agents and socialities: Social and organizational aspects of intelligence. papers from the 2007 AAAI fall symposium* (pp. 57–64).

Iványi, M., Gulyás, L., Bocsi, R., Szemes, G., & Mészáros, R. (2007). *Model Exporation Module.* Agent 2007: Complex Interaction and Social Emergence Conference.

Jacobson, M., & Wilensky, U. (2006). Complex systems in education: Scientific and educational importance and implications for the learning sciences. *Journal of the Learning Sciences*, *15*(1), 11–34.

Janssen, M. A. (2009). Understanding artificial anasazi. *Journal of Artificial Societies and Social Simulation*, *12*(4), 13. Available from `http://jasss.soc.surrey.ac.uk/12/4/13.html`

Jaskowski, W., & Kotlowski, W. (2008). On selecting the best individual in noisy environments. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and Evolutionary Computation* (pp. 961–968). New York, NY, USA: ACM.

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, *9*(1), 3–12.

Jin, Y., & Branke, J. (2005). Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation*, *9*(3), 303–317.

Jones, T., & Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th international conference on genetic algorithms* (pp. 184–192).

Kauffman, S. (1993). *The origins of order: Self organization and selection in evolution.* Oxford University Press, USA.

Kauffman, S., & Levin, S. (1987). Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, *128*(1), 11–45.

Kempe, D., Kleinberg, J., & Tardos, É. (2003). Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 137–146).

Kennedy, J., Eberhart, R., et al. (1995). Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks* (Vol. 4, pp. 1942–1948). Piscataway, NJ: IEEE.

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., & Osawa, E. (1997). Robocup: The robot world cup initiative. In *Proceedings of the first international conference on autonomous agents* (pp. 340–347).

Kleijnen, J. P. C., Sanchez, S. M., Lucas, T. W., & Cioppa, T. M. (2005). A user's guide to the brave new world of designing simulation experiments. *INFORMS Journal on Computing*, *17*(3), 263–289.

Kleijnen, J. P. C., & Wan, J. (2007). Optimization of simulated systems: OptQuest and alternatives. *Simulation Modelling Practice and Theory*, *15*(3), 354–362.

Klein, F., Bourjot, C., & Chevrier, V. (2005). *Dynamical design of experiment with MAS to approximate the behavior of complex systems.* MA4CS.

Klügl, F., Herrler, R., & Fehler, M. (2006). SeSAm: implementation of agent-based simulation using visual programming. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems* (p. 1440).

Kobti, Z., Reynolds, R., & Kohler, T. (2006). The emergence of social network hierarchy using cultural algorithms. *International Journal of Artificial Intelligence Tools*, *15*(6), 963–978.

Kohler, T., Gumerman, G., & Reynolds, R. (2005). Simulating ancient societies. *Scientific American*, *293*(1), 67–73.

Kohler, T., Kresl, J., West, C. van, Carr, E., & Wilshusen, R. (2000). Be there then: a modeling approach to settlement determinants and spatial efficiency among late ancestral pueblo populations of the Mesa Verde region, US southwest. In T. Kohler & G. Gumerman (Eds.), *Dynamics in human and primate societies* (pp. 145–178). New York: Oxford University Press.

Kornhauser, D. (2009). *Representation and parameter space visualization of agent based models.* PhD. thesis proposal.

Kornhauser, D., & Wilensky, U. (2009). *Sweepview [computer software].* Center for Connected Learning and Computer Based Modeling, Northwestern University, Evanston, IL. Available from `http://code.google.com/p/sweepview/`

Koza, J. (1992). *Genetic programming: on the programming of computers by means of natural selection*. The MIT press.

Kratica, J. (1999). Improving performances of the genetic algorithm by caching. *Computers and artificial intelligence*, *18*(3), 271–283.

Kratica, J., Tosic, D., Filipovic, V., Ljubic, I., et al. (2001). Solving the simple plant location problem by genetic algorithm. *RAIRO Operations Research*, *35*(1), 127–142.

Langville, A. N., & Meyer, C. D. (2005, January). A survey of eigenvector methods for web information retrieval. *SIAM Rev.*, *47*, 135–161. Available from `http://portal.acm.org/citation.cfm?id=1055334.1055396`

Leskovec, J., Adamic, L., & Huberman, B. (2007). The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, *1*(1), 5.

Levitan, B., & Kauffman, S. (1995). Adaptive walks with noisy fitness measurements. *Molecular Diversity*, *1*(1), 53–68.

Levy, S., & Wilensky, U. (2008a). Inventing a "Mid Level" to Make Ends Meet: Reasoning between the Levels of Complexity. *Cognition and Instruction*, *26*(1), 1–47.

Levy, S., & Wilensky, U. (2008b). Inventing a mid level to make ends meet: Reasoning between the levels of complexity. *Cognition and Instruction*, *26*(1), 1–47.

Libai, B., Muller, E., & Peres, R. (2009). *The Social Value of Word-of-Mouth Programs: Acceleration versus Acquisition.* (Working Paper)

Luke, S. (2000). *ECJ: A Java-based evolutionary computation and genetic programming system.*

Luke, S., Cioffi-Revilla, C., Panait, L., & Sullivan, K. (2004). Mason: A new multi-agent simulation toolkit. In *Proceedings of the 2004 swarmfest workshop* (Vol. 8).

Ma, T., & Abdulhai, B. (2002). Genetic algorithm-based optimization approach and generic tool for calibrating traffic microscopic simulation parameters. *Transportation Research Record: Journal of the Transportation Research Board*, *1800*(-1), 6–15.

Mahfoud, S. (1995). *Niching methods for genetic algorithms.* Unpublished doctoral dissertation, University of Illinois at Urbana-Champaign.

Maroulis, S., Guimerà, R., Petry, H., Stringer, M., Gomez, L., Amaral, L., et al. (2010). Complex systems view of educational policy research. *Science*, *330*(6000), 38.

Meadows, D. (1997). Places to intervene in a system. *Whole Earth*, *91*, 78–84.

Meadows, D., Behrens, W., Meadows, D., Naill, R., Randers, J., & Zahn, E. (1974). *Dynamics of growth in a finite world.* Wright-Allen Press Cambridge, MA.

Merz, P., & Freisleben, B. (1998). On the effectiveness of evolutionary search in high-dimensional NK-landscapes. In *Proceedings of the 1998 IEEE International Conference*

on Evolutionary Computation (pp. 741–745). IEEE Press.

Midgley, D., Marks, R., & Kunchamwar, D. (2007). Building and assurance of agent-based models: An example and challenge to the field. *Journal of Business Research*, *60*(8), 884–893.

Milgram, S. (1967). The small world problem. *Psychology today*, *2*(1), 60–67.

Miller, J. H. (1998). Active nonlinear tests (ANTs) of complex simulation models. *Management Science*, *44*(6), 820–830.

Minar, N., Burkhart, R., Langton, C., & Askenazi, M. (1996). *The swarm simulation system: A toolkit for building multi-agent simulations.* Santa Fe Institute.

Miner, D. (2010). *A framework for predicting and controlling system-level properties of agent-based models*. Unpublished doctoral dissertation, University of Maryland Baltimore County.

Miner, D., & desJardins, M. (2008). Learning abstract properties of swarm systems. In *Proceedings of the 8th international conference on autonomous agents and multiagent systems.*

Mitchell, M. (2009). *Complexity: A guided tour*. Oxford University Press, USA.

Mitchell, M., Crutchfield, J. P., & Das, R. (1996). Evolving cellular automata with genetic algorithms: A review of recent work. In *Proceedings of the first international conference on evolutionary computation and its applications.* Moscow, Russia.

Mitchell, M., Holland, J., & Forrest, S. (1994). When will a genetic algorithm outperform hill climbing? In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems* (Vol. 6, pp. 51–58). San Mateo, CA: Morgan Kaufmann.

Moreno, J., & Jennings, H. (1938). Statistics of social configurations. *Sociometry*, *1*(3/4), 342–374.

Muhlenbein, H. (1991). Evolution in time and space-the parallel genetic algorithm. In *Foundations of genetic algorithms.*

Narzisi, G., Mysore, V., & Mishra, B. (2006). Multi-objective evolutionary optimization of agent-based models: an application to emergency response planning. *Proceedings of the Second IASTED International Conference on Computational Intelligence*.

Nathan, A., & Barbosa, V. (2008). V-like formations in flocks of artificial birds. *Artificial life*, *14*(2), 179–188.

Newman, M. (2003). The structure and function of complex networks. *SIAM review*, *45*(2), 167–256.

Newman, M. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, *103*(23), 8577.

Newman, M. (2010). *Networks: an introduction*. Oxford University Press.

Newman, M., Barabasi, A., & Watts, D. (2006). *The structure and dynamics of networks*. Princeton University Press.

Nikolai, C., & Madey, G. (2009). Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation*, *12*(2), 2. Available from `http://jasss.soc.surrey.ac.uk/12/2/2.html`

North, M., Howe, T., Collier, N., & Vos, J. (2005). The Repast Simphony runtime system. In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms, ANL/DIS-06-1, co-sponsored by Argonne National Laboratory and The University of Chicago.*

North, M., & Macal, C. (2007). *Managing business complexity: discovering strategic solutions with agent-based modeling and simulation*. Oxford University Press, USA.

Novak, M., & Wilensky, U. (2006). *NetLogo Daisyworld model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from `http://ccl.northwestern.edu/netlogo/models/Daisyworld`

Packard, N. (1988). Adaptation toward the edge of chaos. *Dynamic patterns in complex systems*, *212*.

Page, S. (2010). *Diversity and Complexity.* Princeton University Press. Available from `http://books.google.com/books?id=Mi6zkXss14IC`

Page, S. E. (2008). *The difference: How the power of diversity creates better groups, firms, schools, and societies.* Princeton University Press.

Panait, L., & Luke, S. (2004). Learning ant foraging behaviors. *ALIFE IX: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*.

Panait, L., & Luke, S. (2005). Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, *11*(3), 387–434.

Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas.* Basic Books, Inc. New York, NY, USA.

Parunak, H. V. D., Savit, R., & Riolo, R. L. (1998). Agent-based modeling vs. equation-based modeling: A case study and users' guide. In *Proceedings of the first international workshop on multi-agent systems and agent-based simulation* (pp. 10–25). London, UK: Springer-Verlag.

Penner, D. (2000). Explaining systems: Investigating middle school students' understanding of emergent phenomena. *Journal of Research in Science Teaching*, *37*(8), 784–806.

Perlis, A. (1982, September). Epigrams on programming. *SIGPLAN Notices*, *17*(9), 7–13.

Peterson, R. (1999). Wolf-moose interaction on Isle Royale: the end of natural regulation? *Ecological Applications*, *9*(1), 10–16.

Peterson, R., Page, R., & Dodge, K. (1984). Wolves, moose, and the allometry of population cycles. *Science*, *224*(4655), 1350.

Pfaffmann, J. O., Bousmalis, K., & Colombano, S. (2004). A scouting-inspired evolutionary algorithm. *CEC '04: Congress on Evolutionary Computation, 2004*, *2*.

Pfaffmann, J. O., & Jenkins, J. (2008). Distributed automated experimentation for agent based models. *Proceedings of the Swarmfest 2008 Conference*.

Pfaffmann, J. O., & Zauner, K. P. (2001). Scouting context-sensitive components. *The Third NASA/DoD Workshop on Evolvable Hardware (EH-2001, Long Beach)*, 12–14.

Purcell, K., Rainie, L., Mitchell, A., Rosenstiel, T., & Olmstead, K. (2010). Understanding the participatory news consumer. *Pew Internet and American Life Project*, *1*.

Raia, F. (2005). Students' understanding of complex dynamic systems. *Journal of Geoscience Education*, *53*(3), 297.

Railsback, S., Lytinen, S., & Jackson, S. (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation*, *82*(9), 609.

Rana, S., Whitley, L. D., & Cogswell, R. (1996). Searching in the presence of noise. *Lecture Notes in Computer Science*, *1141*, 198–207.

Rand, W., & Rust, R. (2011). Agent-based modeling in marketing: Guidelines for rigor. *International Journal of Research in Marketing*.

Rand, W., & Sondahl, F. (2004). The El Farol bar problem and computational effort: Why people fail to use bars efficiently. In M. J. North, C. M. Macal, & D. L. Sallach (Eds.), *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence* (pp. 71–86). IL, U.S.A..

Rand, W., & Wilensky, U. (2007). *NetLogo El Farol model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from

http://ccl.northwestern.edu/netlogo/models/ElFarol

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution.* Frommann-Holzboog Verlag, Stuttgart.

Resnick, M. (1994). *Turtles, termites and traffic jams.* MIT Press.

Resnick, M., & Wilensky, U. (1993). Beyond the Deterministic, Centralized Mindsets: New Thinking for New Sciences. In *American educational research association.*

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *Siggraph '87: Proceedings of the 14th annual conference on computer graphics and interactive techniques* (pp. 25–34). New York, NY, USA: ACM.

Reynolds, R., Kobti, Z., Kohler, T., & Yap, L. (2005). Unraveling ancient mysteries: reimagining the past using evolutionary computation in a complex gaming environment. *IEEE Transactions on Evolutionary Computation*, *9*(6), 707.

Rockafellar, R., & Uryasev, S. (2000). Optimization of conditional value-at-risk. *Journal of risk*, *2*, 21–42.

Rogers, A., Tessin, P. von, & Eurobios, U. (2004). Multi-objective calibration for agent-based models. In *5th workshop on agent-based simulation.*

Rosenblueth, A., & Wiener, N. (1945). The role of models in science. *Philosophy of Science*, *12*(4), 316–321.

Rubinstein, R., & Kroese, D. (2004). *The cross-entropy method: a unified approach to combinatorial optimization, monte-carlo simulation, and machine learning.* Springer-Verlag New York Inc.

Ryan, B., & Gross, N. (1943). The diffusion of hybrid seed corn in two Iowa communities. *Rural sociology*, *8*(1), 15–24.

Ryan, C., Collins, J., & Neill, M. (1998). Grammatical evolution: Evolving programs for an arbitrary language. *Genetic Programming*, 83–96.

Sabelli, N. (2006). Complexity, technology, science, and education. *Journal of the Learning Sciences*, *15*(1), 5.

Sanchez, S. M., & Lucas, T. W. (2002). Exploring the world of agent-based simulations: simple models, complex analyses. In *WSC '02: Proceedings of the 34th conference on winter simulation* (pp. 116–126).

Santos, F., Santos, M., & Pacheco, J. (2008). Social diversity promotes the emergence of cooperation in public goods games. *Nature*, *454*(7201), 213–216.

Schelling, T. (1969). Models of segregation. *The American Economic Review*, *59*(2), 488–493.

Schelling, T. (1978). *Micromotives and macrobehavior*. W. W. Norton.

Shaikh, N., Rangaswamy, A., & Balakrishnan, A. (2006). *Modeling the diffusion of innovations using small-world networks* (Tech. Rep.). Working paper. Penn State University.

Shapiro, J. (2001). *Modeling the supply chain*. Duxbury Pacific Grove, California.

Sierra, C., Sabater, J., Augusti, J., & Garcia, P. (2004). SADDE: Social agents design driven by equations. In *Methodologies and software engineering for agent systems*. Kluwer Academic Publishers.

Simon, H. (1973). The organization of complex systems. *Hierarchy theory: The challenge of complex systems*, 1–27.

Skolicki, Z., Arciszewski, T., Houck, M., & Jong, K. D. (2008). Co-evolution of terrorist and security scenarios for water distribution systems. *Advances in Engineering Software*, *39*(10), 801 - 811.

Smith, R., Dike, B., & Stegmann, S. (1995). Fitness inheritance in genetic algorithms. In *Proceedings of the 1995 ACM symposium on applied computing* (pp. 345–350).

Smith, R., & Smith, J. (2001). New methods for tunable, random landscapes. *Foundations of Genetic Algorithms*, *6*, 47–69.

Smith, T., Husbands, P., Layzell, P., & O'Shea, M. (2002). Fitness landscapes and evolvability. *Evolutionary Computation*, *10*(1), 1–34.

Socha, K., & Kisiel-Dorohinicki, M. (2002). Agent-based evolutionary multiobjective optimisation. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), Hawaii USA, IEEE Press* (pp. 109–114).

Sondahl, F., & Rand, W. (2007). Evolution of non-uniform cellular automata using a genetic algorithm: diversity and computation. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and Evolutionary Computation* (pp. 1531–1531). New York, NY, USA: ACM.

Stonedahl, F., Rand, W., & Wilensky, U. (2008a). CrossNet: a framework for crossover with network-based chromosomal representations. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and Evolutionary Computation* (pp. 1057–1064). New York, NY, USA: ACM.

Stonedahl, F., Rand, W., & Wilensky, U. (2008b). Multi-agent learning with a distributed genetic algorithm: Exploring innovation diffusion on networks. In *Proceedings of the ALAMAS+ALAG workshop at AAMAS '08.*

Stonedahl, F., Rand, W., & Wilensky, U. (2010). Evolving viral marketing strategies. In *Proceedings of the 12th annual conference on Genetic and Evolutionary Computation* (pp. 1195–1202). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/1830483.1830701`

Stonedahl, F., & Wilensky, U. (2010a). *BehaviorSearch [computer software].* Center for Connected Learning and Computer Based Modeling, Northwestern University, Evanston, IL. Available from `http://www.behaviorsearch.org/`

Stonedahl, F., & Wilensky, U. (2010b). Evolutionary robustness checking in the Artificial Anasazi model. In *Proceedings of the 2010 AAAI fall symposium on complex adaptive systems.* Available from `http://www.aaai.org/ocs/index.php/FSS/FSS10/paper/view/2181`

Stonedahl, F., & Wilensky, U. (2011). Finding forms of flocking: Evolutionary search in abm parameter-spaces. In T. Bosse, A. Geller, & C. Jonker (Eds.), *Multi-Agent-Based Simulation XI* (Vol. 6532, p. 61-75). Springer Berlin / Heidelberg. Available from `http://dx.doi.org/10.1007/978-3-642-18345-4\_5` (10.1007/978-3-642-18345-4_5)

Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, *11*(4), 341–359.

Tewksbury, D. (2003). What do Americans really want to know? Tracking the behavior of news readers on the Internet. *Journal of Communication*, *53*(4), 694–710.

Tewksbury, D. (2005). The seeds of audience fragmentation: Specialization in the use of online news sites. *Journal of Broadcasting & Electronic Media*, *49*(3), 332–348.

Tisue, S., & Wilensky, U. (2004). NetLogo: Design and implementation of a multi-agent modeling environment. In *Proceedings of Agent 2004* (pp. 7–9).

Toubia, O., Goldenberg, J., & Garcia, R. (2008). *A new approach to modeling the adoption of new products: aggregated diffusion models* (Tech. Rep.). MSI Report.

Ursem, R. (2002). Diversity-guided evolutionary algorithms. *Parallel Problem Solving from Nature (PPSN) VII*, 462–471.

Van Beers, W. C. M., & Kleijnen, J. P. C. (2008). Customized sequential designs for random simulation experiments: Kriging metamodeling and bootstrapping. *European Journal of Operational Research*, *186*(3), 1099–1113.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, *13*(2), 260–269.

Vucetich, J. A., et al. (2011, March). *The Wolves and Moose of Isle Royale project.* Available online: `http://www.isleroyalewolf.org/`.

Vucetich, J. A., & Peterson, R. O. (2009). Wolf and moose dynamics on isle royale. In A. P. Wydeven, T. R. Deelen, & E. J. Heske (Eds.), *Recovery of gray wolves in the great lakes region of the united states* (pp. 35–48). Springer New York. Available from `http://dx.doi.org/10.1007/978-0-387-85952-1_3`

Wakeland, W., Shervais, S., & Raffo, D. (2005). Heuristic optimization as a V&V tool for software process simulation models. *Software Process: Improvement and Practice*, *10*(3).

Waldrop, M. (1992). *Complexity: The emerging science and the edge of order and chaos.* Simon & Schuster.

Wang, J., Dam, G., Yildirim, S., Rand, W., Wilensky, U., & Houk, J. (2008). Reciprocity between the cerebellum and the cerebral cortex: Nonlinear dynamics in microscopic modules for generating voluntary motor commands. *Complexity*, *14*(2).

Wasserman, S., & Faust, K. (1994). *Social network analysis: Methods and applications.* Cambridge University Press.

Watson, A., & Lovelock, J. (1983). Biological homeostasis of the global environment: the parable of Daisyworld. *Tellus B*, *35*.

Watts, D. (2002). A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences of the United States of America*, *99*(9), 5766.

Watts, D. (2004). *Six degrees: The science of a connected age*. WW Norton & Company.

Watts, D., & Dodds, P. (2007). Influentials, networks, and public opinion formation. *Journal of Consumer Research*, *34*(4), 441–458.

Watts, D., & Strogatz, S. (1998). Collective dynamics of 'small-world' networks. *Nature*, *393*(6684), 409–10.

Weinberg, R. (1970). *Computer simulation of a living cell*. Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

Whitley, L. D. (1989). The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the 3rd international conference on genetic algorithms* (pp. 116–123). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Available from `http://portal.acm.org/citation.cfm?id=645512.657257`

Whitley, L. D. (1991). Fundamental principles of deception in genetic search. In G. J. E. Rawlins (Ed.), *Foundations of genetic algorithms* (Vol. 3, pp. 221–241). San Mateo, CA: Morgan Kaufmann.

Whitley, L. D. (1999). A free lunch proof for gray versus binary encodings. In *In* (pp. 726–733). Morgan Kaufmann.

Whitley, L. D., & Kauth, J. (1988). GENITOR: A different genetic algorithm. In *Proceedings of rocky mountain conference on artificial intelligence* (pp. 118–130).

Whitley, L. D., Mathias, K., Rana, S., & Dzubera, J. (1995). Building better test functions. In *Proceedings of the sixth international conference on genetic algorithms* (pp. 239–246).

Wilensky, U. (1997a). *NetLogo Ants model*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from `http://ccl.northwestern.edu/netlogo/models/Ants`

Wilensky, U. (1997b). *NetLogo Fireflies model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from `http://ccl.northwestern.edu/netlogo/models/Fireflies`

Wilensky, U. (1997c). *NetLogo Fire model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from `http://ccl.northwestern.edu/netlogo/models/Fire`

Wilensky, U. (1997d). *NetLogo Segregation model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from `http://ccl.northwestern.edu/netlogo/models/Segregation`

Wilensky, U. (1997e). *NetLogo Wolf Sheep Predation model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from `http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation`

Wilensky, U. (1998). *NetLogo Flocking model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from `http://ccl.northwestern.edu/netlogo/models/Flocking`

Wilensky, U. (1999). *Netlogo.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from `http://ccl.northwestern.edu/netlogo/`

Wilensky, U. (2000, Winter). Modeling Emergent Phenomena with StarLogoT. *@CONCORD.org*.

Wilensky, U. (2001). Modeling nature's emergent patterns with multi-agent languages. In *Proceedings of EuroLogo*.

Wilensky, U. (2004). *NetLogo Heatbugs model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from

http://ccl.northwestern.edu/netlogo/models/Heatbugs

Wilensky, U. (2005). *NetLogo Preferential Attachment model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from http://ccl.northwestern.edu/netlogo/models/PreferentialAttachment

Wilensky, U. (2006). Complex systems and restructuration of scientific disciplines: Implications for learning, analysis of social systems, and educational policy. In J. Kolodner (Chair), C. Bereiter (Discussant), and J.D. Bransford (Discussant) (Ed.), *Annual meeting of the american educational research association.*

Wilensky, U., & Papert, S. (2010, August). Restructurations: Reformulations of knowledge disciplines through new representational forms. In *Proceedings of Constructionism 2010.* Paris, France.

Wilensky, U., & Rand, W. (2003). *NetLogo Ethnocentrism model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from http://ccl.northwestern.edu/netlogo/models/Ethnocentrism

Wilensky, U., & Rand, W. (2007). Making models match: Replicating an agent-based model. *Journal of Artificial Societies and Social Simulation*, *10*(4), 2. Available from http://jasss.soc.surrey.ac.uk/10/4/2.html

Wilensky, U., & Rand, W. (in press). *An introduction to agent-based modeling: Modeling natural, social and engineered complex systems with NetLogo.* Cambridge, MA: MIT Press.

Wilensky, U., & Reisman, K. (1998). Learning biology through constructing and testing computational theories–An embodied modeling approach. In *Second international conference on complex systems* (pp. 171–209).

Wilensky, U., & Reisman, K. (2006). Thinking like a wolf, a sheep, or a firefly: Learning biology through constructing and testing computational theories-an embodied modeling

approach. *Cognition and Instruction*, *24*(2), 171–209.

Wilensky, U., & Resnick, M. (1999). Thinking in Levels: A Dynamic Systems Approach to Making Sense of the World. *Journal of Science Education and Technology*, *8*(1), 3–19.

Wilensky, U., & Shargel, B. (2002). *BehaviorSpace [Computer Software].* Center for Connected Learning and Computer Based Modeling, Northwestern University, Evanston, IL. Available from `http://ccl.northwestern.edu/netlogo/behaviorspace.html`

Wilkerson-Jerde, M., Stonedahl, F., & Wilensky, U. (2010). *NetLogo Flocking Vee Formations model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Available from `http://ccl.northwestern.edu/netlogo/models/FlockingVeeFormations`

Wimsatt, W., et al. (1994). The ontology of complex systems: levels of organization, perspectives, and causal thickets. *Canadian Journal of Philosophy*, *20*, 207–274.

Wolfram, S. (2002). *A new kind of science* (Vol. 1). Champaign, IL, U.S.A.: Wolfram Media.

Wolpert, D., & Macready, W. (1997, Apr). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67-82.

Yahja, A., & Carley, K. (2006). WIZER: Automated model improvement in multi-agent social-network systems. *Coordination of Large-scale Multiagent Systems*, 255.

Zitzler, E., Laumanns, M., Thiele, L., et al. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm. In *Eurogen* (Vol. 3242, pp. 1–21).

Vita

Forrest Stonedahl (né Sondahl) grew up primarily in small-town Idaho before attending Carleton College, presiding over the croquet society there, graduating *summa cum laude*, and then attending graduate school in computer science at Northwestern University after a brief failed career of being a ski bum. His research interests span a variety of subjects, including the areas of artificial intelligence, evolutionary computation, social networks, and multi-agent modeling of complex systems. Stonedahl is a firm believer in interdisciplinary collaboration, and his past projects include applications from archeology to zoology, from linguistics to marketing, and from urban development to materials science. He is also dedicated to improving the quality of computer science education at all academic levels, and takes a particular interest in innovative curricula. Through the excitement of exploring emergent phenomena in multi-agent systems, Stonedahl seeks to inspire a new generation of computer scientists and computational thinkers, who can harness the power of this world's increasingly distributed and decentralized technology for the benefit of tomorrow's society.

For a list of Forrest Stonedahl's publications and other academic accomplishments, please see his website: `http://forrest.stonedahl.com/`