INNOVATION FROM A COMPUTATIONAL SOCIAL SCIENCE PERSPECTIVE: ANALYSES AND MODELS

by

Randy M. Casstevens

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computational Social Science

Committee:

_____ Chair of Committee

_____

_____

_____

_____ Director of Graduate Studies

_____ Director, Krasnow Institute for Advanced Study

Date: _____ Spring Semester 2013
George Mason University
Fairfax, VA

Innovation from a Computational Social Science Perspective: Analyses and Models

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Randy M. Casstevens
Master of Science
North Carolina State University, 2003
Bachelor of Science
North Carolina State University, 2000

Director: Robert Axtell, Professor
Department of Computational Social Science

Spring Semester 2013
George Mason University
Fairfax, VA

Copyright © 2013 by Randy M. Casstevens
All Rights Reserved

Dedication

To my parents, who have always been supportive of my academic pursuits.

To my wife, Erin, who has been my strongest supporter during the entire journey, from the graduate school applications to the dissertation defense; for this, I will always be grateful.

Acknowledgments

I would like to thank my advisor, Robert Axtell, for the wealth of knowledge that he has provided about a broad range of subjects. I also would like to thank my dissertation committee, Kenneth De Jong, Daniel Carr, and William Rand, for all of the valuable suggestions that they have provided over the years. Also, without the foresight of Professor Claudio Cioffi-Revilla, the Department of Computational Social Science would not exist, so I am thankful for all of his efforts in creating a department where interdisciplinary research is appreciated and encouraged.

I gratefully acknowledge the funding support for this research that was provided by the NSF (Award Number 0915657) and the George Mason University Presidential Scholarship. The data analyses in this dissertation would not be possible without the data provided by MathWorks and Google.

For providing feedback on drafts, I would like to thank William Kennedy, Jessica Hughes, Gloria White, and Terry Casstevens. I want to thank all of the faculty and students from the Department of Computational Social Science for the various discussions over the years. A special thanks goes to Karen Underwood, the Academic Programs Coordinator; she has been extremely helpful in ensuring a smooth experience through graduate school.

I would like to thank all of my friends and family for all of their support. My mother, father, brother, sister, and in-laws have always provided boosts of encouragement all along the way. Most importantly, I want to thank my wife, Erin, for all of the love and support.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Abstract

INNOVATION FROM A COMPUTATIONAL SOCIAL SCIENCE PERSPECTIVE: ANALYSES AND MODELS

Randy M. Casstevens, PhD

George Mason University, 2013

Dissertation Director: Robert Axtell

Innovation processes are critical for preserving and improving our standard of living. While innovation has been studied by many disciplines, the focus has been on qualitative measures that are specific to a single technological domain. I adopt a quantitative approach to investigate underlying regularities that generalize across multiple domains. I use a novel approach to better understand the innovation process by combining computational models with empirical data on software development, on one hand, and the evolution of the English lexicon on the other. Innovation can be viewed as the recombination and mutation of existing building blocks. I focus on how building blocks are used to generate innovations. The building blocks are pieces of code (e.g., functions or objects) for the software development data and words for the written language. These data lie at extremes of time scales: innovation occurring over the course of a few days or a week in the case of software while language evolution occurs over decades or centuries. This allows the examination of innovation processes that range from highly-constrained to completely open-ended. Computational methods reinforce the findings from the data analyses and permit exploration of the general features of innovation processes through the construction of abstract models.

The research in this dissertation may be of interest to several types of researchers and

here I hope to guide the reader to the most relevant chapters. For innovation researchers, all chapters may be relevant while those most interested in empirical data analysis should focus on Chapters 2 and 4 and for those interested in how those empirical results are extended with computational models should explore Chapters 3 and 5. For computational social scientists, Chapters 3 and 5 are most relevant because they present the results for the computational models. For software engineering researchers, Chapters 2 and 3 contain the results for the analyses and models of software development. Finally, for linguists, Chapter 4 contains the results for the analyses on word usage and growth.

Chapter 1: Introduction

1.1 Motivation

Innovation is a driving force in the economies [1] across the world [2]. With a better understanding of how innovation progresses, insights on how to accelerate the rate of innovation may be discovered. Agent-based models are a relatively new technique for modeling social phenomena [3], suitable for representing social processes on a variety of scales [4]. This flexibility of scale makes agent-based modeling a valuable tool for the study of innovation. Innovation occurs at every level of society from the global community [5] [6] coming together to address a global concern (e.g., international health [7] [8] [9] and climate change [10] [11]) to individuals searching for better solutions to their problems [12]. These agent-based models should be motivated by the real world, so empirical data analysis drives the design of the models.

When an inventor creates an innovation, it is done using component parts from existing products or processes [13]. Therefore, an evolutionary approach will be used in the modeling approach. This means that a new innovation will be derived from a population of existing goods. Also, there will be a selection process that chooses which goods make it to the next generation. There has been much research by the evolutionary computation community [14] [15] about these types of systems and its insightful experience will be used whenever possible.

This research focuses on the collaborative side of innovation. As Arthur nicely stated:

> "[A] single practitioner's new projects typically contain little that is novel. But many different designers acting in parallel produce novel solutions." [13, pg. 101]

Technological progress does not happen in isolation by a single individual. As discussed later, Sawyer's work [16] disputes the "myth of the solitary genius" as the primary source of innovation. In additional to collaboration between individuals, innovation occurs with the help of existing technologies as building blocks and apparatuses to aid in the creation of something new [13]. This idea of recombining existing building blocks to create something new will be a persistent thread throughout this dissertation. With this focus on the building blocks of innovation, many different aspects of the innovation process can be quantitatively explored, including the problem-solvers' contributions, the problems being solved, and the solutions that are created.

This dissertation explores two extreme types of innovation. The first is characterized by a short-term process that has well-defined goals. The second is characterized by a long-term process that has ever-changing and open-ended goals. Furthermore, another important difference is that in the first type of innovation, the introduction of an innovation does not influence the fitness of other innovations, but in the second type of innovation, the system is co-evolutionary, thereby the introduction of a new innovation can change the fitness of other innovations. As a result of this, the first type of innovation will have static fitness landscapes and the second will have dynamic fitness landscapes.

The breakdown of fitness landscapes into static and dynamic provides the structure of the dissertation. Chapters 2 and 3 explore the analysis and modeling of systems with static fitness landscapes, respectively, while Chapters 4 and 5 explore dynamic fitness landscapes. Chapter 6 concludes the dissertation by summarizing results and proposing directions for future work.

1.2 Research Questions

Many disciplines have studied innovation, but generally focus on qualitative aspects of the innovation process. Here I attempt to find quantitative approaches to the study of innovation so that they can be applied to multiple technological domains. Therefore, the central question being asked in this dissertation is:

- What aspects of the innovation process can be studied quantitatively?

Since innovations are built from component parts, the quantitative study of innovation requires an understanding of how building blocks are assembled, which leads to another essential question:

- How are building blocks used during innovation processes?

This dissertation focuses on building blocks because they are important and little has been done to understand the dynamics of how building blocks are used in real-world innovation processes. Every innovation of any complexity is made up of component parts. Examples of products without component parts include simple objects like a paperweight, which is of little interest to researchers of innovation. This examination of building blocks has three distinct aspects to study:

- The problem being solved
- The problem-solvers (i.e. the innovators)
- The solutions generated (i.e. the innovations)

Empirical data sets allows the exploration of all of these aspects, although not all of these aspects can be studied for both data sets used in this dissertation.

Two empirical data sets are used to answer these questions: MATLAB programming contest [17] data and word frequency data from the Google Books [18] project. The building blocks are pieces of code (i.e., functions) for the programming contest data and words for the word frequency data. Empirical data analysis and computational models complement each other in this dissertation. With data analysis, the behavior of actual human problem-solvers can be quantified and studied with statistical techniques. However, data analysis may not provide a complete picture of the phenomenon under study. With computational models, additional scenarios can be created and systematically studied. Furthermore, computational models provide the ability to study the mechanisms behind the innovation processes.

In the remainder of this chapter, a survey of various important theories about innovation are introduced. By borrowing from the existing theories, a theory of innovation is developed for this dissertation and will be discussed later in this chapter. These theories of innovation will be used to generate specific research questions regarding building blocks that can be tested with empirical data analysis and/or computational models.

1.3 Group Problem Solving

In this section, I provide an overview of research dealing with how groups solve problems. Surowiecki's book, "The Wisdom of Crowds" [19], describes the recipe for using a crowd to make wise decisions. The other two books discussed in this section, "Group Genius" by Keith Sawyer [16] and "The Rise of the Creative Class" by Richard Florida [20], center around fostering creativity. Creativity will not play a key role in this dissertation, but the research in this area provides insight into group problem solving.

1.3.1 "The Wisdom of Crowds" by James Surowiecki

Surowiecki [19] explains how groups of individuals can consistently perform better at a task than any single individual in the crowd. He provides many examples of collective wisdom and explains that there are two core characteristics of the crowd for it to be wise: diversity and independence. Diversity of opinion ensures that each individual is using the private information available to them. The works of Scott Page [21] [22] and Richard Florida [20] have also stressed the importance of diversity. Independence of opinion ensures that each individual does not become overly influenced by those around them. Both diversity and independence are important because they prevent the errors of the individuals in the crowd from becoming correlated [19].

The wise crowd must also be decentralized. This means that the individuals in the crowd need to be able to use their local knowledge and specialize. The final component of a wise crowd is having a mechanism to aggregate the thoughts of the individuals. Without a method of aggregation, there is no way to generate the solution to the problem [19]. The

programming contest data, discussed in detail in Chapters 2 and 3, allows the study of a unique way of aggregating solutions from a crowd.

1.3.2 "Group Genius" by Keith Sawyer

In his book, Sawyer [16] disputes the "myth of the solitary genius," where innovations are created by a single individual working in isolation. His book focuses on group creativity with much of his data analysis coming from creative performances, including jazz ensembles and improv groups. Others have also suggested that creativity is not a solitary endeavor. In Richard Ogle's book entitled "Smart World" [23], he emphasizes the role of networks of individuals and ideas in the creative process. Sawyer based his research on the research of Mihaly Csikszentmihalyi [24] and developed what he calls "group flow" where groups can coalesce [16].

In his book entitled "Flow: The Psychology of Optimal Experience" [24], Csikszentmihalyi describes the state of 'flow' as one when someone is totally absorbed in the task at hand. Csikszentmihalyi [24] believed that 'flow' is an essential ingredient for creativity and provides happiness to the individual experiencing it. 'Flow' is more likely to occur when performing a task with four characteristics:

1. The required skill matches the individual's talents.
2. There is a clear goal.
3. There is immediate feedback.
4. The individual's full concentration is on the task [24].

Sawyer's research [16] centers on creativity and focuses on aspects that tend to be difficult to quantify. Throughout this dissertation, group collaboration will play a central theme but, unlike Sawyer's research [16], the entities under study (i.e., the building blocks) are easily quantifiable.

1.3.3 "The Rise of the Creative Class" by Richard Florida

Florida's book [20] explores how the economy of the United States is moving away from manufacturing to an economy centered on creativity. He refers to the 30 percent of the American workforce that creates "meaningful new forms" [pg. 68] as the creative class and this group is growing in importance in the American economy. He goes on to describe community characteristics that help facilitate creativity, including: thick labor markets, urban lifestyles, social interactions, increased diversity, and the authenticity, identity, and quality of the location [20]. His research adds motivation to the research discussed in this dissertation because Florida expects that the rise of the creative economy will be the next economic revolution in the United States.

1.4 Theories of the Evolution of Technology

Technological change has been studied many times and there are many varying theories about technological progress. This section will review three theories about the evolution of technology that have influenced my ideas about innovation. The following two subsections will review three books: Arthur's "The Nature of Technology"[13], Basalla's "The Evolution of Technology" [25], and Kelly's "What Technology Wants" [26]. These books provide a compelling theory of technological evolution and support their theory with many examples. The final subsection will discuss the theory explained in the book "The Structure of Scientific Revolutions" by Kuhn [27]. Kuhn's theory focuses on the progress of scientific endeavors, but provided interesting insights for the study of innovation that should be noted.

1.4.1 "The Nature of Technology" by W. Brian Arthur

Arthur's theory of the evolution of technology centers on the idea that novel innovations come from new combinations of existing technologies [13]. Others have made this same argument, including Schumpeter [28]. As these new innovations appear, made of new combinations of existing components, the complexity of the innovations grow over time. Since

a new innovation grows out of existing elements, which also were built of existing elements, technologies have a recursive, hierarchical structure. This hierarchy leads to a modular structure of technology [13]. Simon also found that complex systems commonly use hierarchical systems for their architecture [29]. In addition to combination and recursiveness, a technology must harness some natural phenomena [13]. Combination and recursiveness that lead to modularity are the most important principles from Arthur's book for this dissertation, while the use of natural phenomena is not as directly applicable.

This theory falls very much in line with my thoughts about the evolution of technology. However, Arthur's focus on the growth of technology complexity caused by the combination of existing technologies does not include a need for parsimony. Arthur mentions the cycle of increased complexity followed by simplification, but importance was not placed on the idea [13]. In the software development domain, the simplification phase is important as maintainability of the software becomes more expensive in time and money as complexity grows. This cyclic pattern of growth in complexity followed by reduction of complexity is not as apparent in some domains as software development.

Arthur's theory also notes that technologies are not thrown together randomly. As discussed later, computational models in this dissertation randomly combine elements together as a baseline for comparison. Moreover, some interesting patterns can be recreated with these simple randomly behaving agents. Therefore, as a general rule, models begin simple and complexity is only added when necessary.

1.4.2 "The Evolution of Technology" by George Basalla

Basalla [25] cautiously compares the evolution metaphor of the "world of the made" versus the "world of the born" because the first is a result of purposeful human activity and the latter is a result of random natural processes. Another difference between technological and biological evolution is that the forces influencing the creation process of new innovations are also influencing the selection process. Also, he notes that humans have chosen to use "excessively complex" technologies to satisfy basic necessities and that biological necessity

is not the primary reason for technological innovation [25].

Basalla's theory of technological evolution consists of four broad concepts: diversity, continuity, novelty, and selection [25]. There is a vast diversity of technologies and they are continuing to grow. Continuity means that a new innovation proceeds from an antecedent. In selection, technologies compete for survival with the aid of humans. The selection process is influenced by economic forces, but "by no means is the main actor" [25, pg. 144]. I found the lack of reliance on economic forces particularly compelling, especially since some of my data comes from problem-solvers who are not receiving any direct monetary compensation.

The role of imagination about technology is noted as a rich source of innovative ideas and how this is particularly true in Western society [25]. He also states that there has been a long-held claim that technology grows at an exponential rate. Importance is placed on cultural and social views about technology. For example, Judeo-Christian beliefs promoted technological innovation and gave individuals in the West an advantage in innovation. Basalla also stresses the importance of trial and error when creating a new innovation [25]. Trial and error search is also found in evolutionary computation, which will be discussed more later in this chapter.

1.4.3 "What Technology Wants" by Kevin Kelly

Kevin Kelly is the cofounder of Wired magazine and has some interesting insights into the development of technology in his book "What Technology Wants" [26]. One point that Kelly makes that is most relevant to this dissertation is that the development of technologies is inevitable. He argues that for any evolutionary process, the large-scale characteristics of the products of evolution are determined by the nature of the landscape. Therefore, if the evolutionary process could be run again, then it would produce the same macrostructure. However, the small details of the product would not necessary be the same. Kelly uses the carving of a river as an example. The overall location of the river is determined by the structure of the landscape and water would find the same path if the evolutionary process was run multiple times [26].

Since the development of a particular technology is inevitable, it is natural to find the phenomenon of multiple individuals discovering the same technology at approximately the same time. Once the underlying technologies have been discovered, then the subsequent discoveries are ripe for the picking and many times happen simultaneously by independent discoverers [26]. Kelly points out many examples of simultaneous discovery of ideas and products [26].

The idea that technologies are inevitable has mixed implications for this dissertation. As a computational modeler, the existence of inevitable technologies is an excellent property because it implies that models will (or should) be robust to small changes in initial conditions. However, from a complex systems perspective, inevitable technologies implies that innovators are simply performing a search task over the technological landscape. Therefore, it is the landscape that matters most in the innovation process and not the multitude of interactions between innovators. Nevertheless, Kelly [26] may just be assuming that there will be interactions between innovators, because, as noted earlier by Arthur [13] and Sawyer [16], novel solutions are produced by many, not a single individual.

While Kelly [26] claims that the macrostructure of the evolution of technology is primarily shaped by the technological landscape, his full theory of innovation depends on three forces, which are borrowed from Stephen Jay Gould's work [30] on biological evolution. Gould's three forces are "structural," "historical," and "functional" [30], while Kelly labels the last force "intentional" [26]. The first force, as noted earlier, is the inevitable force that is dictated by the structure of technology. The second force, the historical force, is the contingent aspect of evolution where what happened in the past matters. The third force, the intentional force, is society's collective decisions of which technologies to use. Kelly sees the third force as intentional actions by individuals and is differentiated from blind natural selection in biology. Unlike Basalla, who was cautious about using the evolution metaphor to discuss the evolution of technology, Kelly uses it with gusto and even in the literal sense when discussing the three forces of evolution:

> "If the technium is indeed the extended acceleration of the evolution of life, it

should be governed by the same three forces." [26, pg. 182]

The 'technium' is the term coined by Kelly to refer to the whole system of technological evolution.

1.4.4 "The Structure of Scientific Revolutions" by Thomas S. Kuhn

The first edition of Kuhn's classic book [27] dates to 1962 and focuses on progress in the scientific community. Before a paradigm is agreed upon in a scientific discipline, there is a preparadigmatic period that is characterized by various approaches competing for acceptance by a community of scientists. The most promising approach is selected by the community as the best way to go about doing science. The paradigm defines the important discoveries to build future research on and the questions that should be the scientists' focus during future research. Therefore, it is critical that the paradigm not only define unprecedented insights but also provide unresolved questions for future scientists to work on [27].

A paradigm shift is the change from a older paradigm to a new one by a scientific community. These paradigm shifts may take many years as researchers whose careers have depended on the old paradigm are resistant to change to the new paradigm. Younger members of the scientific community are more likely to be developers and proponents of the new paradigm because they have not invested a great deal of time and energy in the old paradigm [27]. Varying degrees of acceptance of a new idea is also an important concern for the diffusion of innovation research [31].

Kuhn [27] places importance on the use of textbooks in the scientific community as a way to define the language, important discoveries, and open questions of a paradigm. Also, there is the tendency of authors of textbooks to view the history of science as a linear progression from one breakthrough to the next. Kuhn points out that these progressions may not be a smooth transition from one paradigm to the next, but rather the history of science includes disjoint revolutions or paradigm shifts. However, the textbook authors may smooth out the irregularities of the history of science and only choose to highlight historical

discoveries that can be reformulated in light of the current paradigm [27].

1.5 Important Theories about Technological Progress

There are other important theoretical works about technological progress that will be useful to review. These theories fit in nicely within the theories of technological evolution discussed in the previous section. The theories are the linked S-curve theory of innovation [32], Schumpeter's theory of creative destruction [33], and Schmookler's theory about the source of innovation [34]. The most relevant portions of these theories are briefly discussed.

1.5.1 S-shaped Performance Curves

The S-shaped curve has been proposed as a pattern of technological innovation because improvements are smaller in the initial phases of the product when the number of users are small and the innovation is still becoming understood [35]. As the technology is better understood and there are more users, the performance curve grows, which is eventually followed by a slowing period where the product matures and improvements become more difficult to find [35].

1.5.2 Linked S-curves

This linked S-curve theory [32] extends upon the previous section and consists of recurring S-shaped performance curves. When the technological improvements in one technology become exhausted, another technology takes over the performance curve (see Figure 1.1) [32].

Exponential growth in performance of a technology can extend across multiple S-curves as the dominant technological paradigm changes over time. Koh and Magee [36] show empirical evidence of how linked S-curves can be responsible for the exponential growth across multiple technology paradigms (multiple S-curves). For example, they show that the pattern of exponential growth in computing power found in Moore's law is much older than

the transistor [36]. Using historical data about computational speed, including the machine calculator, early vacuum tube computer, transistor computer, and modern day computer, they show that the exponential growth in computation described by Moore's law is a trend that goes back more than 100 years. They also find this same pattern in other types of technologies including data transportation and storage technologies [36]. Kurzweil [37] also describes the transition from one paradigm to the next in computing power and calls our current computing technologies the "Fifth Paradigm" of computing. Kurzweil also predicts that the sixth paradigm will be three-dimensional computing technologies at the molecular level [37].

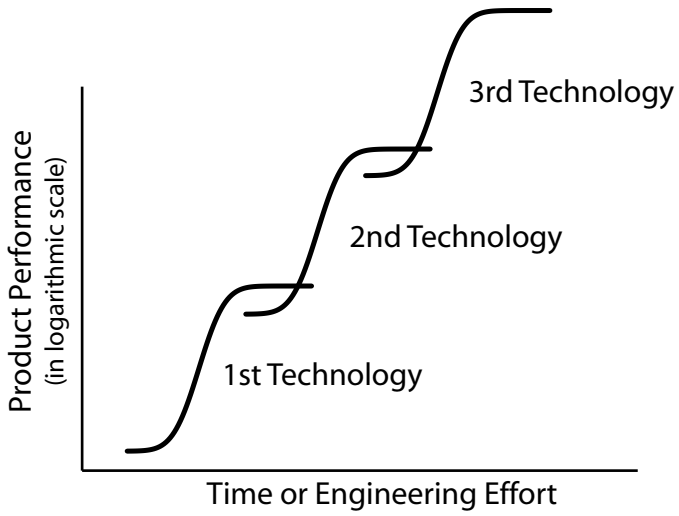


Figure 1.1: Linked S-Curve Theory (from [32] and [36])

At its core, the linked S-curve theory shows how many short-term processes can produce a long-term trend. The dichotomy of short-term and long-term processes have been recognized in other social science fields. For example, Cioffi-Revilla [38] proposes a theory for political development where many 'fast' processes (i.e., short-term) of collective action

compose a 'slow' process of long-term growth in political complexity.

1.5.3 Creative Destruction

Schumpeter [33] notes that capitalism is an evolutionary process that can never be stationary. Also, innovation was very closely tied to Schumpeter's thoughts about capitalism:

> "The fundamental impulse that sets and keeps the capitalist engine in motion comes from the new consumers' goods, the new methods of production or transportation, the new markets, the new forms of industrial organization that capitalist enterprise creates." [33, pg. 83]

The process of creative destruction consists of new products and processes coming into being and destroying the older existing technologies.

1.5.4 Punctuated Equilibria

The original idea of punctuated equilibria comes from Eldredge and Gould [39]. Punctuated equilibria started in biological evolutionary theory by theorizing that species did not evolve by a gradual process, but rather periods of "homeostatic equilibria" were disturbed by rare events of rapid change. Eldredge and Gould [39] propose that this process caused speciation. This same theory has been proposed for the evolution of technology, where a product can remain unchanged for a long period, but there are episodic events of rapid change [40] [41].

1.5.5 Scientific Supply versus Market Demand

When studying the source of innovation, it is important to consider the motivations of the inventors. Do new innovations come about due to new scientific knowledge that makes a new innovation possible? Or does market demand for a product come before the new innovation? As Basalla said:

> "The crucial question is whether inventions are stimulated by the push of the growing supply of knowledge or by the pull of the increasing demands of the marketplace." [25, pg. 113]

Schmookler [34] believed that the market demand was the strongest driver of innovation. He used patent data to provide evidence for this theory [42]. However, Dosi [43] discusses how innovations follow "technological trajectories" that guide future improvements. Certainly, both scientific supply of knowledge and market demand are working together to create new innovations [44] and to distinguish between the two is difficult [45]. Furthermore, as momentum grows on one side, it can encourage momentum in the other. However, economic considerations are not the only reason for innovation, but this debate is still an interesting one.

1.6 Models of the Evolution of Technology

The previous two sections provide excellent examples of theories about technological change, but did not focus on developing models that could be tested with empirical data. This section will focus on two evolutionary models of technological progress. The models are discussed in the two books: Nelson and Winter's "An Evolutionary Theory of Economic Change" [46] and Frenken's "Innovation, Evolution and Complexity Theory" [47].

1.6.1 "An Evolutionary Theory of Economic Change" by Richard Nelson and Sidney Winter

Nelson and Winter [46] recognize the successes of neoclassical economic theory with the common assumptions that it uses, including perfect information, two commodities, and static equilibrium. However, these assumptions can be limiting in that they do not account for technological change during the temporal duration of the model. They propose a simulation approach that incorporates Lamarckian evolutionary theory. In the model, many firms are producing the same product, which is analogous to the gross national product (GNP) (the GNP data and other data from Solow's paper [48] is used for comparison). The production function of a firm is defined by the firm's production technique (defined by two coefficients) and the amount of stock available. Each of the firms can choose to search for a

better production technique or imitate another firm. If the firm chooses to look for a new technique, then a probability distribution is used to generate the new technique. In either case, the firm only accepts the new production technique if it improves production. After calibrating their simulation model with empirical data, the authors generate aggregate time series of variables that are consistent with historical economic data. They are also able to connect microeconomic variables with macro outcomes. Nelson and Winter's model was able to match aggregate economic values that summarize economic growth seen in the first half of the twentieth century in the United States [46].

Nelson and Winter [46] also make note that technological progress can be continual, but sometimes happens very rapidly, which has also been observed in the theories of innovation discussed earlier. In Nelson and Winter's model, there is also a constant set of possibilities for the technology of the production function [46]. Therefore, their model was searching within a set of preexisting possibilities. Nelson and Winter note that the term "search" does not make sense when referring to a totally new technology, but dismiss the difference between these two types of innovation by saying:

> "But for the purpose of our evolutionary modeling, the distinction here is one of semantics not substance." [46, pg. 210]

However, I believe there is a substantial difference between these two types of innovation. In the one type with only preexisting possibilities, there will be a limit on how much progress can be made, and with enough time a search algorithm will be able to find this limitation, but in the more open-ended type of innovation, the possibilities may be endless. These two types of innovation provide the overall structure of this dissertation and will be discussed later in this chapter.

1.6.2 "Innovation, Evolution and Complexity Theory" by Koen Frenken

Like Nelson and Winter [46], Frenken [47] goes beyond a theoretical framework of technological progress and includes a model. Frenken makes use of the NK landscape developed by Kauffman [49] and the generalized NK landscape introduced by Altenberg [50] [51] [52].

The NK landscape will be used in my model of innovation, so it will be discussed later in this chapter.

Frenken [47] models a product's life cycle based on Williamson's [53] stages, which are the explorative stage, the development stage, and the mature stage. In the explorative stage, product innovation is high and volume is low. During the development stage, standardization causes a drop in product innovation, but innovation of the production of the product is high. Also, the volume of production grows rapidly during this stage. In the mature stage, both product innovation and process innovation are low. The product technological innovation remains low until a new product replaces it and the cycle starts over again [47].

Frenken's model [47] introduces two types of innovation: product and process innovation. This recognizes that not only the product is undergoing innovation, but how it is produced is also being developed. Frenken's model also includes the idea of technology trajectory, where an innovation is not totally 'blind', but where there is a sense of which parts of the innovation should be changed or remain the same. This knowledge is used to confine the search in the design space [47].

Entropy and mutual information are used by Frenken [47] to explore technological innovation. Entropy is an indicator of the amount of randomness in the distribution, where a uniform distribution has maximum entropy. Frenken [47] uses the entropy of a type of technology as an inverse indicator of whether or not there is a dominant design. Mutual information gives an indicator of how much a random variable tells about another random variable. The mutual information metric is an indicator of the amount of differentiation in a particular type of technology. Using data from steam engines, aircraft, helicopters, and portable computers, Frenken [47] shows how entropy and mutual information can be used to study different evolution trajectories.

1.7 Other Related Evolutionary Models

This section will focus on evolutionary models that are relevant to my model and especially those models that have been used in the social sciences. These evolutionary models mainly come from biology and economics. Evolutionary biology has been exploring evolution the longest and provides valuable insight in the process of evolution. In economics, a subfield called evolutionary economics has emerged to address these types of models.

1.7.1 Evolutionary Biology

There are differences between biological and technological evolution. For one, Lamarckism is common in the evolution of innovations, but Lamarckism is only found to be true in very limited ways in biological evolution [30]. Lamarckian inheritance (also known as "soft" inheritance) occurs when a parent passes characteristics to offspring that were acquired during the parent's lifetime. The only evidence within biology of Lamarckian inheritance is within microorganisms. For example, Cairns et al. [54] provide evidence of this type of Lamarckian inheritance in E. coli bacteria. While Lamarckism is a limited force within biological evolution, the transfer of characteristics from one innovation to another is widespread in technological evolution.

There are models from biological evolution that are useful in the study of technological evolution. The following sections will briefly review these models.

Kauffman's NK Model

Kauffman's NK model [55] was originally designed to model biology, but was also used by Kauffman to explore technological evolution. This model is interesting for the study of technological evolution because the NK landscape is one where the ruggedness of the landscape can be systemically modified using the K parameter. In the model, N represents the number of elements within each entity (i.e., individual) in the system. This could be the number of genes, amino acids, or some other characteristic of an entity, and there are a number of functions, F, that are influenced by these elements. Each element makes a

contribution to the fitness of the individual along with K other parts [55].

The ruggedness of the landscape is tunable by changing the value of K, where landscapes with higher values of K are more rugged [49]. For example, when K = 0, then each element is independent of the others, so there is a single peak in the fitness landscape. When K = N-1, every element is dependent on every other element, which means that the fitness landscape is completely random and uncorrelated. The most interesting NK landscapes lie between these two extremes where neighboring locations on the fitness landscape are partially correlated [55].

Altenberg's Generalized NK Model

Altenberg's generalized NK model [52] is based on Kauffman's model [49]. Just as before, there are N elements and F functions, but in this model they are not necessarily the same value as in Kauffman's original formulation. Altenberg's model does not have a universal K value. Each gene can affect any number of functions, and each function can be influenced by any number genes [50] [51] [52].

Altenberg [51] uses his generalized NK model to study evolutionary dynamics. In this model, a new gene (or element) is added periodically to the genome and this new gene is assigned a random pleiotropy [51]. The pleiotropy is the number of functions that the new gene influences. Pleiotropy could also be described as the number of connections in the genotype-phenotype map. In his 'constructional selection' scheme, the new gene is only accepted if it improves the total fitness of the individual. Altenberg [51] finds that as the model progresses only genes with a low pleiotropy are added to the genome. Basically, after an initial period of stabilization, the core of the genome is set and only peripheral functions are able to be modified [51]. The generalized NK landscape provides more freedom to experiment with the interactions between compositional elements and functional fitness values.

1.7.2 Evolutionary Economics

Economics has made many contributions to the use of evolutionary processes in models. The combination of evolutionary techniques and economics has been explored in a relatively new field called evolutionary economics. This dissertation will not focus on economic issues, although it is worth reviewing some of the related models.

Economic Models of Production

Auerswald and his collaborators [56] were dissatisfied with the microeconomic foundations of macroeconomic models of production. They provide a description of the production plan by adding an engineering recipe. The NK model, discussed earlier, was used as inspiration for their production recipe model. The model includes one input, one output, and constant returns [56]. Their model is able to recreate the essence of 'learning by doing' that was described by Kenneth Arrow [57] and Wright's law [58], which states that the reduction in production cost falls as a power law with respect to the cumulative production [56][1].

Zero-Intelligence Traders

In this model, an artificial market of zero-intelligence traders chose random prices to buy or sell a resource [59]. Zero-intelligence agents did not seek to maximize profits and had no intelligence or memory. However, the artificial market is still able to have high allocative efficiency [59]. The zero-intelligence agents provide a baseline for comparison and Gode and Sunder [60] used it to show the usefulness of market rules. The market rules that increase efficiency include traders only taking profitable trades, the traders can negotiate profitable trades, and extramarginal traders do not displace intramarginal traders [60]. Gode and Sunder state that "if the trading rules are 'smart,' the traders need not be" [60, pg. 623].

Gode and Sunder's model [59] [60] shows that the neoclassical economic assumption of fully rational agents is unnecessary. In economics, a fully rational agent has unlimited

[1] Wright's law was originally formulated as $l_t = a\,(Y_{t-1})^{-1/3}$, where Y_{t-1} is the cumulative production and l_t is the unit cost at time t. Due to further empirical research since Wright, Auerswald and collaborators used a generalized form of the law ($l_t = a\,(Y_{t-1})^{-b}$) [56].

computational resources and access to global information. The goal of a fully rational agent is solely to maximize the given utility function. Simon was an advocate of replacing fully rational agents with bounded rational ones [29]. He suggested that these types of extensive calculations with large amounts of information are not realistic for the human decision maker. Furthermore, since there is no difference between the real world and the agent's perception of it, then there is no need to study the agent's perception of the real world [61]. One simple form of bounded rationality described by Simon was coined as 'satisficing.' Simon described 'satisficing' as using the first solution found that met a minimum criteria [62] [63]. This is quite similar to the approach used by Gode and Sunder [59] [60].

In the computational models presented in this dissertation, bounded rational agents are used. Furthermore, simplicity in rationality is favored over complicated cognitive mechanisms. This establishes a baseline of comparison and reduces the risk of creating confounding results.

Economic Models using Genetic Programming

Genetic programming has been advocated as a way to include bounded rationality within an agent-based model [64] [65]. When using genetic programming as a technique for bounded rationality, each agent has a population of programs as mental models [64]. An agent can update a program's fitness based on observations from the environment. Also, the learning mechanism is flexible since if an agent's selection criteria change, then another program can become the favored one for use as the agent's mental model [64].

Chen [66] use a similar notion by treating the economic agents as "collection of decisions rules" (term from Lucas [67]). The agent's preferences guide the evolution of the collection of decision rules. The genetic programming approach allows for the growth of agent (or solution) complexity over time [66]. Another paper by Chen [68] proposes using genetic programming as a way to generate modularity using automatically defined terminals (ADTs). Chen and Chie [69] use the genetic programming approach to generate commodities and the agent's preferences for those commodities. Each commodity is represented by a genetic

programming tree and the fitness of the commodity is based on how well it matched the agent's preferences, also represented by a genetic programming tree [69].

1.8 My Theory of the Evolution of Technology

No single element of my theory of the evolution of technology is new. Much like technology itself, the theory is novel in that it is a new combination of existing elements. This theory has been greatly influenced by the theories discussed above.

The main mechanism for my evolution of technology consists of novel combinations of existing elements, or building blocks, to create new innovations. This is much like the theory proposed by Arthur [13], but where Arthur focuses on this mechanism creating greater complexity, I also explore how greater complexity can hinder future innovation. I believe the regulation of complexity to favor parsimony over unnecessary complexity is crucial for the maintainability and expense of a new product, especially in the software development domain, which is one of the domains that I draw empirical data. Furthermore, software development is a rare domain where innovation complexity is observable and quantifiable.

Another common observation in the evolution of technology is the idea of relative stability followed by periods of rapid change [13] [25]. This means that there may be a conflict between the view of technological progress from short-term and long-term perspectives. Therefore, in the next section, I will discuss how innovation processes are separated into two groups for this dissertation.

1.9 My Approach to Innovation Research

My approach to innovation research is influenced by the linked S-curve theory of innovation. As discussed earlier, the S-shaped curve of product performance has been proposed as a pattern of technological innovation [35]. The linked S-curve theory extends this idea and consists of recurring S-shaped performance curves, where a new innovation continues making performance improvements as improvements in the older technology become exhausted [32].

This type of transition from one type of technology to another has been referred to as a 'paradigm shift' [27] [37]. Exponential growth in performance of a technology can extend across multiple S-curves as the dominant technological paradigm changes over time (for empirical evidence see [36] and [37]).

Using the linked S-curve theory as motivation, the empirical data analysis and agent-based models in my dissertation are broken up into two types of innovation. The first type deals with a single S-shaped curve where innovation occurs over relatively short time periods and has well-defined goals. The second type deals with multiple S-shaped curves where innovation occurs over long time periods and has ever-changing and open-ended goals. The first type of innovation is constrained to a single technological paradigm, while the second type could have multiple paradigm shifts.

1.9.1 Empirical Data Analysis

The inclusion of empirical data is an important component of this dissertation and allows for detailed examination of how building blocks are used during the innovation process. There are two sources of data analyzed in this dissertation. The first, from the software development domain, allows analysis of a short-term innovation process (one week) that has well-defined goals (a single S-curve). The second comes from word frequency data that allows analysis of a long-term innovation process (one century) that has open-ended goals (multiple S-curves).

These two domains are interesting case studies in their own right, but I believe that the results from the data analysis will generalize to other domains. As Arthur states:

> "Physically, a jet engine and a computer program are very different things. One is a set of material parts, the other a set of logical instructions. But each has the same structure. Each is an arrangement of connected building blocks that consists of a central assembly that carries out a base principle, along with other assemblies or component systems that interact to support this." [13, pg. 35]

A brief description of the data used is given below, although more details are provided about the empirical data in their respective chapters.

Software Development Data

Data from software developers participating in MATLAB programming contests are used to study the first type of innovation (a single S-curve). The software development process is an excellent domain for the study of innovation. It is a self-documenting system that provides a detailed view of an innovation process with many of the historical artifacts available for examination. The MATLAB programming contests are different from standard contests because once a solution has been submitted, it can be viewed, modified, and reused by other contestants. This allows for collaboration within the competition. Data from twenty-three contests were gathered and analyzed. The contests occurred from December 1998 to March 2012 and there were roughly two contests per year. This data includes every software submission made during the contest (averaging nearly 2,000 submissions per contest). This allows for a detailed examination of the evolution of problem solutions for each of the twenty-three contests. Measures used to study the programming contests include characteristics of the problem being solved (e.g., modularity, size, and complexity), the problem-solvers (e.g., individuals' contributions and social network analysis), and the products of the problem-solving process (e.g., changes in the use of sub-modules over time).

Software development provides an illustration of how the recursive, hierarchical structure of innovations that Arthur mentions in his book [13] can develop. Programming commands are combined to form functions, which are further aggregated to create the software application, thereby creating a hierarchical structure. For this dissertation, functions are the building blocks under study.

There has been much research done on the software development process of the open source community. The open source community is an excellent domain for innovation research because the artifacts are available for study, which is not generally the case for commercial software. However, much of this previous research focuses on how to make the

development process better and not necessarily on how the development process occurs, which is the focus of this dissertation. When appropriate, previous research about the open source community are noted.

Word Frequency Data

Data from the Google Books project are used to study the second type of innovation (innovation over multiple S-curves). The data used for this dissertation includes the word frequency count from over 600,000 books that were published during the 20th century (for more about the Google Books data see [70]). For each year of this 100-year period, word frequency counts were used from approximately 6,174 books per year. For this data, words are the building blocks under study.

The Google Books data gives insight into a truly long-running and open-ended innovation process, which is exactly what is wanted. However, due to the open-ended nature of the data, little can be said about the characteristics of the problem being solved. Also, little is known about the innovators (i.e., the authors of the books) and there is no way of tying words and phrases to the author who wrote them. Therefore, the focus of the analysis is the products of the innovation process (the frequency of words over time).

1.9.2 Computational Models

Another important component of my approach to studying innovation is the use of computational models. Just as there are two types of data analysis (a single S-curve and multiple S-curves), there are two types of models. The first type of computational models studies innovation processes with well-defined goals, and there is a single S-shaped performance curve. These models are based on techniques from evolutionary computation (genetic algorithms). The second type of computational models study innovation processes with open-ended goals, and there are multiple S-shaped performance curves. Creating an open-ended evolutionary process in a computer is difficult, but results are promising. The central goal of all of these computational models is to better understand how building blocks are combined to create

new innovations. Furthermore, the models provide a framework to explore scenarios where there are no empirical data.

The computational models were kept simple intentionally to avoid confounding results. It also illuminates which phenomena are easy to generate and which are hard. When looking for a mechanism for a phenomenon, it is best to follow Occam's Razor and choose the most parsimonious mechanism with the fewest assumptions. This thought is succinctly expressed by a quote commonly attributed to Albert Einstein:

> "Everything should be made as simple as possible, but not simpler."

Before introducing the models in the subsequent chapters, it is important to briefly discuss some of the underlying techniques and terms that are used in the models and why they are included in the model.

Evolutionary Computation

As mentioned earlier, evolutionary computation provided the framework for the models of innovation for a single S-curve. Standard evolutionary computation techniques (genetic algorithm) are used as a baseline for comparison with data from human problem-solvers. The study of building blocks has a long history in the field of evolutionary computation. The importance of building blocks in evolutionary computation goes back to the origins of the field with John Holland's discussions of it [71] [72].

The 'agents' in evolutionary computation are not cognitively sophisticated, but use selection, variation, and reproduction to solve problems. In evolutionary computation, an initial population of problem solutions are randomly generated and are allowed to evolve to generate better solutions. If a pattern from the empirical data appears with this type of model, then it shows that the pattern may not be a product of humans' cognitive ability, but rather a product of the evolutionary process.

In evolutionary computation, a population of solutions reproduce and are selected to create a balance between exploration of unknown regions and exploitation of known regions

of the solution space. Evolutionary computation has a rich history of how to evolve a population of entities using reproduction and selection. The field of evolutionary computation shares many characteristics with the process of innovation. In both cases, a heterogeneous population of solutions or innovations is competing for survival. The individuals with a higher fitness are more likely to survive, while the individuals with lower fitness are more likely to be eliminated. This characteristic is also true of the process of innovation, where the most useful innovations are more likely to permeate society than the innovations of little usefulness.

Agent-Based Modeling

Most times in evolutionary computation, the fitness of a solution is straightforward because the fitness values are easily ordered and comparison between two values is trivial. However, for an open-ended, co-evolutionary innovation process, comparison of solutions is not trivial. For example, when comparing two innovations, the fitness of the innovations may depend on the observer. Some individuals may find some innovations more desirable than other individuals. Therefore, innovations evolve in a system where individuals have idiosyncratic objectives. Moreover, some individuals may be more accepting of new ideas than others. Therefore, the model of open-ended innovation (multiple S-curves) used agent-based modeling. Agent-based models make it quite natural to include heterogeneous agents.

Heterogeneous Agents

Creating agents with idiosyncratic preferences is an important benefit of agent-based models [73] [4]. When modeling innovation, heterogeneous agents are important because all individuals do not behave the same or have the same information. Therefore, information is lost if only the aggregate characteristics of the population are considered rather than the entire population. For example, Rogers wrote extensively on how individuals adopt a new innovation at different times and described various classifications of individuals (innovators, early adopters, early majority, late majority, and laggards) in terms of when they adopted

an innovation [31].

Adoption rate is not the only important type of individual heterogeneity. As seen earlier in Nelson and Winter's work [46], the firms being modeled had heterogeneous production techniques, probabilities for imitation of others, and propensity to explore for new innovations. Example uses of heterogeneity in the agent-based model presented in this dissertation include: variation in the solutions being used by the agents and the agents' evaluations of those solutions. This heterogeneity allows there to be diversity in the available building blocks and this is crucial when creating a model that relies on recombination.

Growth in Complexity

As a technology progresses, the number of component elements tends to grow [74]. Arthur explains why technology grows in complexity:

> "Complexity tends to increase as functions and modifications are added to a system to break through limitations, handle exceptional circumstances, or adapt to a world itself more complex." [75, pg. 144]

Arthur also notes that eventually a simplifying concept will be discovered and simplicity will replace the complexity [75]. This regulation of complexity seems particularly important in the software development domain because as complexity grows, maintainability becomes more difficult.

Agent Interactions

Agent-based models easily allow for interactions between agents, which can be difficult in other types of models (e.g., mathematical models) [73] [4]. As noted earlier, novel innovations are created by groups and not by a solitary genius [16] [13]. Therefore, when modeling innovation, it is critical to allow the innovators to interact with one another.

1.9.3 Data Visualization

The third and final major component of my approach to innovation research is the use of data visualization. Various types of standard data visualizations (e.g., scatter plots, box plots, bar charts, etc.) are used throughout this dissertation. These standard visualizations provide understanding of the data, although sometimes modifications were required to better illustrate the topic of discussion. Furthermore, at times, a custom visualization was required to make the point more clear. Also, in the visualizations used here, color schemes developed by Cynthia Brewer were used (http://colorbrewer2.org/). These color schemes not only make the visualization more aesthetically appealing, but also aid in the ease of perceiving the information [76] [77] [78][2].

1.10 Discussion

Innovation is studied by many different fields, consequently it is important to note what aspects of innovation will not or only slightly be covered by this dissertation. Also, when referring to the process of innovation, I am referring to the act of problem solving and the process by which solutions change over time. I am not referring to an innovation as something that must have a major change in the lifestyle of the users.

The idea of creativity is outside the realm of this dissertation. I am going to follow the lead of Basalla:

> "The findings of psychological research into the wellsprings of creativity are not included, because that material is not immediately pertinent to the theory of technological evolution." [25, pg. 65]

Furthermore, creativity is something that is difficult to quantify and I will focus on patterns that are quantifiable. Certainly, creativity is an important part of the study of innovation, but requires the modeling of individuals' mental processes. This dissertation will focus on

[2]The research by Brewer and colleagues have centered on creating color schemes for maps, but I find their color schemes quite useful for all types of visualizations.

the population of innovations and aggregate patterns of these innovations, not on how a single innovation is created.

Also, this work will not focus on the history of technology. There are many accounts of the history of various types of technology. An overview of the evolution of technology with many historical examples can be found in other sources, I especially recommend Basalla's [25] and Arthur's books [13].

I want to briefly give an overview of the remainder of this dissertation. In Chapter 2, empirical data analysis of the MATLAB programming contest data is discussed. This data is from an innovation process that occurs over a relatively short time period and has well defined goals (a single S-curve). In Chapter 3, computational models of this first type of innovation (a single S-curve) are introduced. In Chapters 2 and 3, the innovation landscape is static and there are no major paradigm shifts. Chapters 4 and 5 follow the same pattern as Chapters 2 and 3, where a data analysis chapter is followed by a modeling chapter. Chapter 4 discusses the data analysis of the word frequency data from the Google Books project, which is followed by discussion of an agent-based model in Chapter 5. In Chapters 4 and 5, the innovation process is occurring over long time scales, where the problems are open-ended and ever-changing. The landscape is dynamic and dependent on what other solutions are in the environment, thereby it is coevolutionary. To relate the organization of the chapters with Figure 1.1, Chapters 2 and 3 discuss the innovation of a single S-curve, while Chapters 4 and 5 explore innovation across multiple S-curves.

My research provides a tangible way of studying collaborative problem solving by providing quantitative measures of the innovation process under study. The data sets used in this dissertation lie at extremes; one demonstrates an innovation process that occurs over the course of a week and the other over the course of a century. This provides a broad range of innovation processes to study, from processes with well-defined goals to ones that are open-ended and ever-changing. Chapter 6 concludes this dissertation by summarizing the lessons learned from this research and proposes some future research directions.

Chapter 2: Analyses of Static Fitness Landscapes

In this chapter, I investigate innovation on a static fitness landscape. Specifically, I study the progressive improvement in solutions submitted to programming contests, held over relatively short periods of time. This problem domain is able to teach us how individual solutions build on previous solutions in order to achieve progressively better performance. Given the highly-constrained character of such domains, with respect to both the character of the problems being solved and the time permitted for solutions to be developed, I can safely ignore the complexities of long-run, open-ended, dynamic innovation processes in which the problems being solved by new inventions are themselves evolving. With regard to the S-curve model of innovation discussed in the last chapter, the innovations investigated in this chapter represent progress along a single S-curve. In Chapters 4 and 5, I will investigate overlapping S-curves associated with long run technological progress.

2.1 Software Development

This chapter uses empirical data from the software development domain. There are several reasons why this is an excellent domain for the study of innovation. 1) The historical record is many times pristinely preserved. 2) Many attributes are easily quantifiable (e.g. lines of code, execution time, code complexity). 3) The author of the code modification is also in the historical record, thereby allowing the study of individuals' contribution levels. 4) Software engineering is an important domain in its own right. 5) Software engineering is many times a collaborative activity and provides an excellent example of group problem solving. 6) Lastly and most importantly, I see software engineering as the act of systematically solving problems with computers. Hopefully, the insights found from the study of innovation in software engineering will provide a clearer picture for other types of innovation.

2.2 Programming Contest Data

Data from MATLAB programming contests [17] were gathered and analyzed.[1] This contest data has shown to be an excellent way to explore collaborative problem-solving. The MATLAB programming contests are different from traditional contests in that a submission is not necessarily written by a single individual. The contestants could test their problem solutions with a small set of example inputs provided by the contest administrators, but to get a more comprehensive evaluation of their solution, the contestant must submit their solution to the contest administrators. Once submitted, the program is scored and made available to all contest participants. The solution's score is a combination of the solution's performance, complexity, and CPU runtime (see Table 2.2 for more details). Other contestants could use any previous submission(s) as the starting point of their solution [17] [81]. Moreover, a contestant can change a single line of code in another contestant's submission and take the lead in the contest [82]. This is referred to as "tweaking".

Previous analyses of the MATLAB programming contests include the work of Ned Gulley, the lead architect of the MATLAB programming contests, and Karim Lakhani [83]. They focused on the factors that influenced individual's performance and the collective value of submissions to other participants [83]. In their work, they found that novel recombinations of other participant's code were a statistically significant predictor of the submission being the best-so-far solution [83]. Therefore, Gulley and Lakhani [83] found that collaboration in the form of novel recombination was an important contributor to innovation in the programming contests. Here, I focus on other aspects of the programming contests that provide further insights into the innovation process.

2.2.1 Contest Mechanics

The contests generally last for seven days and for the first two days, the submissions are not visible to other participants. On day two of the contest, all submissions, even ones

[1] An earlier version of this analysis was presented at the 2nd Annual Complexity in Business Conference [79] and the 7th European Meeting on Applied Evolutionary Economics [80]. I would like to thank the participants of these two conferences for valuable feedback.

submitted during the first two days, are visible by all participants. Also, to encourage early submissions, there was an early bird prize during the competition [17]. Since I am exploring collaborative problem-solving for this data analysis, the period where participants can see each other's submission (the last five days) is of most interest and thereby, only this data was used for the analysis.[2] The period where submissions are not visible (the first two days) is treated as a period to create an initial set of solutions. This is analogous to creating an initial population in evolutionary computation.

This type of competition is much more useful in studying collaboration compared to the standard contests where the contestants work in isolation during the entire duration of the contest. Since any participant can use previously submitted programs as the starting point for their submission, this allows for collaboration within a competitive contest. Moreover, these contests have an interesting mix of collaboration and competition reminiscent to the open source software movement [81].

Data from twenty-three programming contests were gathered from the MATLAB programming contest website [17]. The data was gathered using a web scraping program that gathered scoring, author, and rank information along with the MATLAB code that was submitted. Table 2.1 includes summary information about the programming contests including the number of submissions, participants, score leaders, and improvements for each of the programming contests. This table only includes information about the submissions that where submitted during the last five days of the competition when participants could see others' submissions. The number of score leaders is a count of anyone that was in the lead during the competition. The number of improvements is a count of the number of times an improvement to the best-so-far solution was submitted (regardless of the size of the improvement).

[2]For some of the contests, it was unclear when the different phases of the contest started, so to be consistent across contests, I used all submissions that were submitted two days after the first contest submission.

Table 2.1: Summary information about the MATLAB programming contests

Contest Name	Date	Number of Submissions	Number of Participants	Number of Score Leaders	Number of Improvements
Tiles	Apr 2012	1274	35	17	141
Vines	Nov 2011	1081	57	18	108
Crossword	Mar 2011	1439	55	15	74
Sailing Home	Nov 2010	2731	62	29	216
Sensor	Apr 2010	3522	70	17	256
Color Bridge	Nov 2009	1867	54	17	109
Army Ants	Nov 2008	1850	43	3	7
Wiring	Apr 2008	3206	59	22	162
Gene Splicing	Nov 2007	2172	67	21	158
Peg Solitaire	May 2007	2959	86	25	165
Blackbox	Nov 2006	4409	105	19	61
Blockbuster	Apr 2006	4593	102	34	191
Sudoku	Nov 2005	1870	90	25	129
Ants	May 2005	1702	107	14	78
Moving Furniture	Nov 2004	1155	54	24	164
Gerrymandering	Apr 2004	1803	92	37	181
Trucking Freight	Apr 2003	1079	71	34	165
Protein Folding	Nov 2002	1465	109	26	105
Molecule	May 2002	801	89	14	58
Mastermind	Sep 2001	345	55	11	29
Gene Sequencing	Mar 2000	234	41	5	12
Mars Surveyor	June 1999	1351	44	12	79
Binpack	Dec 1998	876	101	18	61

2.2.2 Submission Scoring Criteria

Since the first MATLAB programming contest, the scoring criteria has changed a bit over the years.[3] Table 2.2 gives a summary of the scoring criteria used for each of the contests. Figure 2.1 shows the relationship between the innovation measures and the problem characteristics. While these variables will be discussed in detail later in this chapter, what this figure shows is that the change in scoring criteria does not introduce a systemic bias into the analysis. Therefore, all contests will be treated as the same regardless of the variation in scoring criteria. Moreover, the scoring criteria always includes the solution quality and computational speed, which from the contest rules are the most important factors in the score [17].

The contest administrators did not post the scoring function for the most recent contests (the ones that use result quality, execution time, complexity, and node count) on the contest rules. However, the following equation was used for the peg solitaire contest where result quality, execution time, and complexity was used in the scoring [17]:

$$score = k_1 * result + k_2 * e^{(k_3 * runtime)} + k_4 * max(complexity - 10, 0) \tag{2.1}$$

The smaller the score the better the submission. For all of the equations in this section, the values of the k_i parameters were unpublished, although these parameter values can be estimated. Using the estimated values for the k_i parameters, I calculated the percent of the score that was due to each of the three scoring components: result quality, execution time, and complexity. On average for all scored submissions for the peg solitaire contest, result quality accounted for 93.6%, execution time for 6.3%, and complexity less than .1% of the score values. This gives a clear indication that the contest administrators, and presumedly the contest participants, placed priority on result quality and execution time. Also, note that there is a penalty to include a function that has a complexity that is greater than ten, but if all functions have a complexity of ten or less than there is no penalty. Keeping the

[3]This section is included for completeness and to justify my decision of which data I chose to analyze. Most readers can skip this section without losing the overall message of this chapter.

complexity of code at a reasonable level is a good programming practice that makes the code easier to understand and modify. Also, as noted in the contest rules, the node count criteria was added to motivate participants to remove unused code from their submissions [17].

The following equation was used for several of the contests that only used result quality and execution time in scoring [17]:

$$score = k_1 * result + k_2 * e^{(k_3 * runtime)} \tag{2.2}$$

Two of the earliest contests (mars surveyor and gene sequencing) used this equation to generate scores based on result quality and execution time [17]:

$$score = k_1 * result + k_2 * runtime \tag{2.3}$$

After reviewing the various scoring criteria, it is not expected that the differences in scoring will introduce confounding results into the analysis. Moreover, the contest administrators introduction of new criteria during the course of the contests was motivated to get the participants to write simple and elegant solutions without any unnecessary code [17]. Therefore, all twenty-three contests was used in the analysis. However, for those interested, I placed some results in Appendix A.1 that only include the first thirteen contests, where only result quality and execution time is used in scoring.[4]

[4]The results using only the first thirteen contests tell a similar story as the arguments that I make in this chapter.

Table 2.2: Summary of scoring criteria for each of the MATLAB programming contests.

Contest Name	Result Quality	Speed	Cyclomatic Complexity	Node Count
Tiles	X	X	X	X
Vines	X	X	X	X
Crossword	X	X	X	X
Sailing Home	X	X	X	X
Sensor	X	X	X	X
Color Bridge	X	X	X	X
Army Ants	X	X	X	X
Wiring	X	X	X	X
Gene Splicing	X	X	X	
Peg Solitaire	X	X	X	
Blackbox	X	X		
Blockbuster	X	X		
Sudoku	X	X		
Ants	X	X		
Moving Furniture	X	X		
Gerrymandering	X	X		
Trucking Freight	X	X		
Protein Folding	X	X		
Molecule	X	X		
Mastermind	X	X		
Gene Sequencing	X	X		
Mars Surveyor	X	X		
Binpack	X	X		

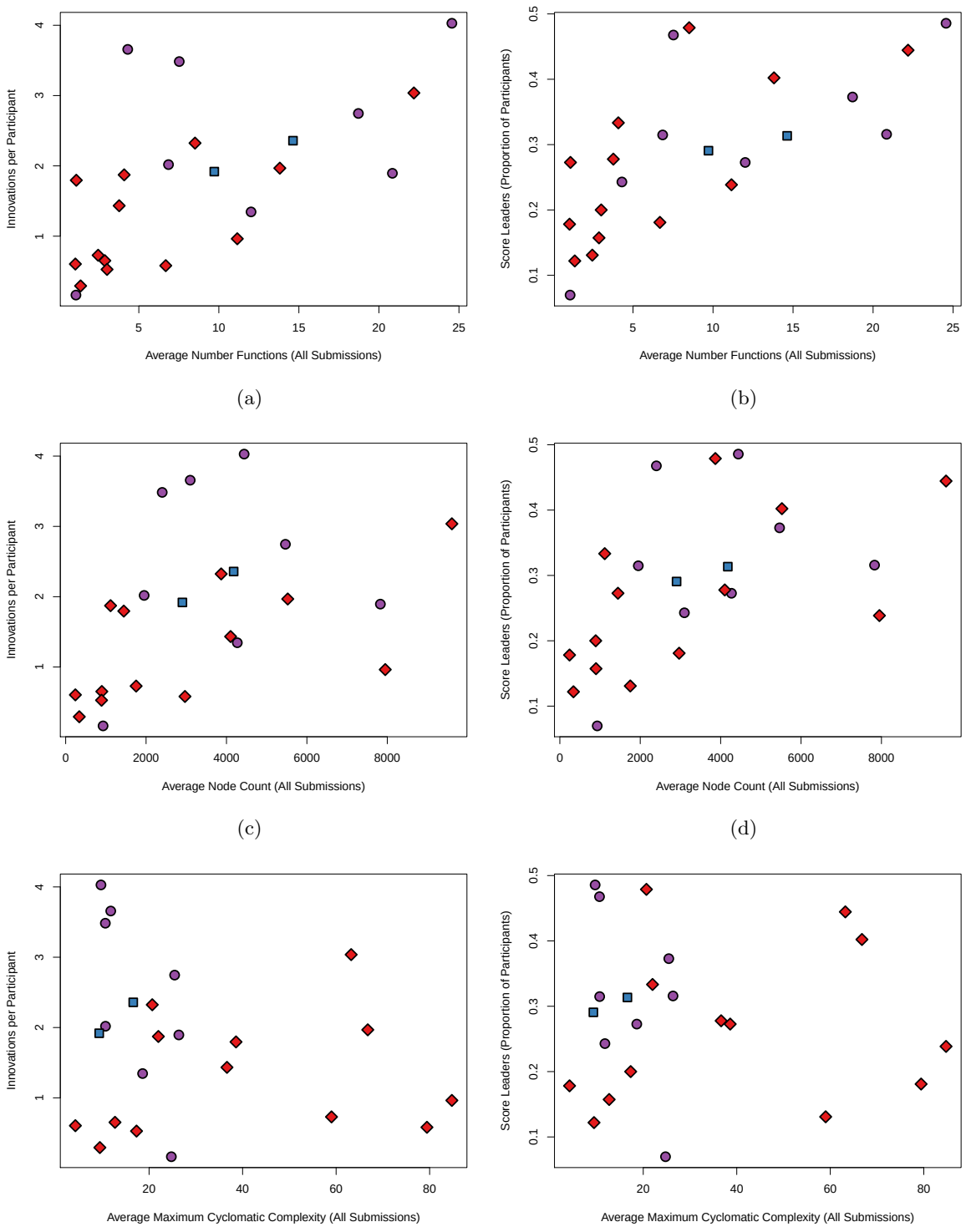


Figure 2.1: Relationship between innovation variables and problem characteristics using the different scoring criteria. The red diamonds represent the first thirteen programming contests where only result quality and execution speed was used to judge a submission. The blue squares represent two contests that added cyclomatic complexity to the scoring criteria. Finally, the purple circles represent the eight most recent programming contests which added node count.

2.3 Research Questions

Question 1: How does the characteristics of the problem being solved influence the innovation process?

Question 2: Are there any similarities between the programming contest data and the open source software community?

Question 3: Does the programming contests show evidence of creative destruction?

The first question focuses on the problem being solved, the second on the problem-solvers, and the third on the products of the problem-solving process. In the remainder of this chapter, each of these questions are answered using various quantitative approaches.

2.4 Types of Analysis

Three aspects of the MATLAB programming contest data were analyzed: the problem being solved, the problem-solvers, and the problem solutions. In the first analysis, when studying the problem being solved, linear regression was used to investigate the effect of three problem characteristics (i.e., size, modularity, and complexity) on the innovation process. With a better understanding of how the characteristics of the problem being solved influences the rate of innovation, obstacles like Brooks's Law may be avoided. Brooks's Law states that adding more software developers to a late project will make it later [84].

The second analysis explored the problem-solvers. In this analysis, the distributions of developers per project and projects per developer was investigated to see if they follow a heavy tailed distribution and possibly a power law[5]. Power laws have been found for the developers per project and projects per developer distributions in the open source community [85]. The programming contest data provided an opportunity to examine if this type of pattern could emerge in programming contests that occur over a relatively short time period, for approximately one week. Moreover, finding these types of distributions in

[5]For readers not familiar with power laws, an overview of power laws, including the two formulations of them, is discussed in Appendix E.

the programming contest data may help draw parallels between the relatively simple programming contests and the highly complex community of open source software developers. Studying a simpler system that resembles the open source community may shed light onto previously unknown aspects of open source software development.

In the last analysis, changes in the solution population were recorded to examine if there is evidence of creative destruction. In creative destruction, the introduction of a new innovation causes other products to no longer be used and was credited by Schumpeter as the "fundamental impulse that sets and keeps the capitalist engine in motion" [33, pg. 83]. By calculating the diversity of the submission population before and after a best-so-far submission, the reduction in submission diversity was investigated and the narrowing of the problem-solvers focus was measured. For comparison, the same operation was also performed for the submissions that were not best-so-far submissions. A statistical analysis was performed to see if the better submissions caused a narrowing of focus by the problem-solvers and thereby, creative destruction.

2.5 Analysis of the Problem Being Solved

Characteristics of the problem being solved were analyzed using ordinary least squares (OLS) regression. The regression analysis focuses on the influence of problem modularity, size, and complexity on software developers ability to improve contest solutions. Data from twenty-three programming contests were analyzed. The analysis provides a quantitative way of comparing the importance of problem characteristics on the innovation process. The following dependent and independent variables were used in the linear regression.

2.5.1 Measures of Innovation

The dependent variables are the number of innovations per participant and the percentage of participants that were innovators. An innovation is simply anything that improved upon the best solution so far and an innovator is anyone that submitted one of those best-so-far solutions. These measures of innovation were chosen to allow for comparison across contests.

Each contests have a different number of participants, so these innovation measures take this factor into account. Furthermore, submitted solutions are easily compared within a contest, but difficult between contests and thereby, solution improvements are difficult to compare across all contests. Thus, the count of innovations during the contest provide a consistent measure of innovation that is applicable to all contests.

2.5.2 Characteristics of the Problem Being Solved

The independent variables are the size, modularity and complexity of the problem being solved by the participants. The problem's modularity, size and complexity are not characteristics that are directly observable from the problem statement. Therefore, the participants' submissions were used to calculate an estimate of these characteristics. Size was measured by the average number of nodes in the parse tree of all of the submissions. The size of the parse tree is a better indicator of program length than the number of lines of code. Number of lines of code as a measure of program length can be misleading due to differences in programming style that do not affect the execution of the program. Modularity was measured by the average number of sub-functions of all the submissions. Complexity was measured using the cyclomatic complexity, also known as McCabe complexity. Cyclomatic complexity measures the number of independent linear pathways through the code [86]. MATLAB calculates the cyclomatic complexity for each function and a single submission may include multiple functions, so the maximum complexity for each submission was averaged to obtain the complexity of the problem being solved. Only submissions that did not generate an error during execution and received a score were used in these calculations.

One problem with using modularization as a predictor of innovation is the notion that more modules simply means longer programs which is the real reason for the increased amount of innovation. This problem is addressed with this analysis by determining which factor (i.e. problem size, modularity, or complexity) explains the most variation in the measures of innovation.

The results discussed here used the average modularity, size, and complexity for all of

the contest submissions. Separate analyses were also done for the problem characteristics of the winning submissions and the average of the best-so-far submissions. These other variables were highly correlated with the ones reported here and did not change the nature of the results, so they were not included.

2.5.3 Excluded Data

Currently, the MATLAB programming contest website contains information about twenty-five programming contests. Two of the contests were not included in the analysis. One was excluded because it was a visualization contest and thereby, the submissions do not have a objective way of being scored and compared. The other excluded contest scored submissions based on the number of characters in the submitted program that correctly converted the input into the desired output. Unlike the rest of the contests, solution quality and execution time were not a component of a submission's score. Since creating sub-functions required more characters in the code, the solutions were not broken down into sub-functions. This is substantially different from the other contests and thereby, excluded from the analysis. The data from the remaining twenty-three MATLAB programming contests were analyzed using linear regression.

2.5.4 Regression Results

Tables 2.3 and 2.4 display the results of the regression analysis. Table 2.3 used innovations per participant and Table 2.4 used percent of participants that were innovators as the dependent variable. Each row of the tables shows the results of the regression for each of the independent variables: average number of functions, average node count, and average maximum complexity. The modularity variable (average number of functions) and the size variable (average node count) were both significant predictor for both innovation variables. However, the complexity variable was not a significant predictor for either of the innovation variables. The effect size (measured by R-squared) of the modularity variable was approximately double that of the size variable. This provides quantitative evidence

that problem modularization is a better predictor of innovation than problem size and both problem modularization and size are better predictors than problem complexity.

Table 2.3: Regression results from the 23 MATLAB contests with innovations per participant as the dependent variable.

	Dependent Variable: Innovations per Participant		
Independent Variables:			
Average Number Functions			
Coefficient	**0.094**		
p-value	**0.002**		
Average Node Count			
Coefficient		**1.9E-04**	
p-value		**0.041**	
Average Maximum Complexity			
Coefficient			-0.009
p-value			0.365
Intercept	0.927	1.120	2.026
R-squared	0.380	0.185	0.039
Adjusted R-squared	0.351	0.146	-0.007
F-statistic	12.880	4.754	0.856

Note: Values in **bold** are significant at the 5% level.

Table 2.4: Regression results from the 23 MATLAB contests with percentage of participants that were innovators as the dependent variable.

	Dependent Variable:		
Independent Variables:	Percent Innovators		
Average Number Functions			
Coefficient	**0.011**		
p-value	$<$ **0.001**		
Average Node Count			
Coefficient		**2.5E-05**	
p-value		**0.009**	
Average Maximum Complexity			
Coefficient			-2.1E-04
p-value			0.848
Intercept	0.187	0.202	0.292
R-squared	0.470	0.282	0.002
Adjusted R-squared	0.445	0.248	-0.046
F-statistic	18.640	8.244	0.038

Note: Values in **bold** are significant at the 5% level.

2.6 Analysis of the Problem-Solvers

In open source software, the distributions of developers per project and projects per developer follow a power law distribution [85]. For this chapter, a similar analysis was performed for the programming contests. Identifying the developers was straightforward, but establishing 'projects' within a contest was not quite as clear. During the contest, the participants were given the opportunity of specifying another submission as the basis of their code. However, this information was incomplete, as this information was not always given. Therefore, 'projects' were established directly from the submissions themselves.

2.6.1 Comparison of Code Submissions

In order to group similar submissions to make a 'project', the parse tree of the functions in each submission was generated and compared with all other functions submitted during the contest (see Algorithm 2.1 for details). User defined names of functions and variables

were not used during the comparison because those are easily changed to obfuscate the code without affecting the execution of the code. Also, if $function_i$ is labeled as function type x and $function_j$ is later found to be of type x, then $function_k$ can be labeled as type x if it is within the threshold T of either $function_i$ or $function_j$. This allows the function type to represent an entire evolutionary trajectory, not simply a cluster of functions that are similar to one another. The method from Baxter et al. [87] was used for comparing parse trees:

$$similarity(tree_i, tree_j) = \frac{S_{i,j}}{mean(N_i, N_j)} \tag{2.4}$$

$S_{i,j}$ is the number of nodes that are shared between the two trees and N_i is the total number of nodes in $tree_i$. While $tree_i$ is the parse tree for the i^{th} function. This method also requires a threshold parameter T that specifies when two parse trees are considered the same, which will be discussed later in this section.

Algorithm 2.1 Using Function Parse Trees to Build the Submission Lineage.

$ClassifiedTrees \leftarrow \emptyset$
for all $function_i$ **do**
  $tree_i \leftarrow parse\_tree(function_i)$
  $maxSimilarity \leftarrow \arg\max_j(similarity(tree_i, tree_j \in ClassifiedTrees))$
  if $maxSimilarity > T$ **then**
    Classify $tree_i$ as the same as $tree_{max_j}$
  else
    Create a new tree classification
  end if
  $ClassifiedTrees \leftarrow ClassifiedTrees \cup tree_i$
end for

The order of the branches in the parse tree were not used in the comparison. If a developer changed a single line of code at the beginning of a function, then it would shift all of the branches of the parse tree down. Therefore, a strict comparison requiring the

branches to appear in the original order would classify the small change to the function as a totally new function. Consequently, a strict ordering of branches was not required during the comparison process. Moreover, by not being concerned with the order of the branches in the parse tree, it helped reduce the execution time of this already long running process. The comparison between two functions ranged between zero and one and measured the proportion of parse tree nodes that were in both functions. These function comparisons allowed the source code to be classified into a set of function types. A unique set of function types were considered a 'project'. This approach allows the lineage of the submissions to be reconstructed directly from the submissions themselves.

Determining the Similarity Threshold

Determining the threshold for similarity (T) includes conflicting motivations.[6] On one side, a stricter threshold (a higher T) is better because the different approaches to solving the problem can be identified. On the other side, a looser threshold (a lower T) is better because it preserves more of the lineage of the evolving solutions. While a stricter threshold can introduce a different function type when there was an actual lineage, a looser threshold can insist there was a lineage when there was none.

All is not lost because the quality of a threshold can be compared with the empirical data. When submitting a solution, the participant had the option of specifying another submission that their submission was 'based on'. However, from examining the empirical data, there are a few systematic reporting errors with the 'based on' variable. First, participants may not report that their submission was based on another submission even if it was. Second, participants can only specify one submission as the 'based on' submission. Therefore, if they combined multiple submissions into one, then that information cannot be provided by the participant. Third, participants could say their submission was based on their previous submission even if they borrowed heavily from someone else's submission.

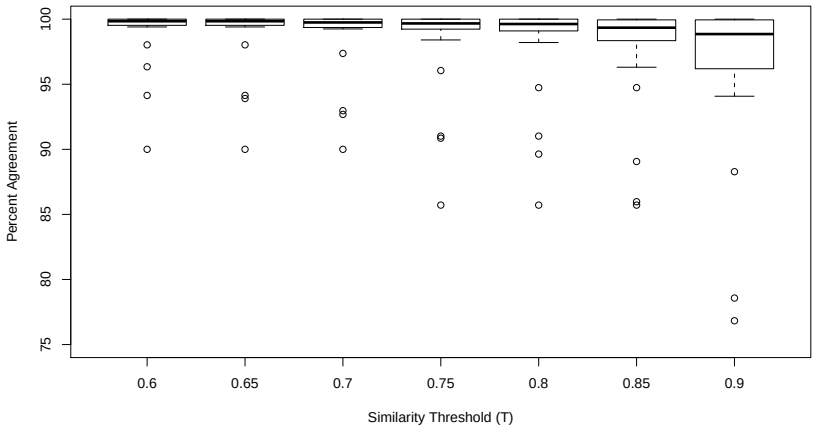
With these reporting errors in mind, one situation where it seems that the 'based on'

[6]This section is included for completeness and to justify my decision of similarity threshold T chosen. Most readers can skip this section without losing the overall message of this chapter.

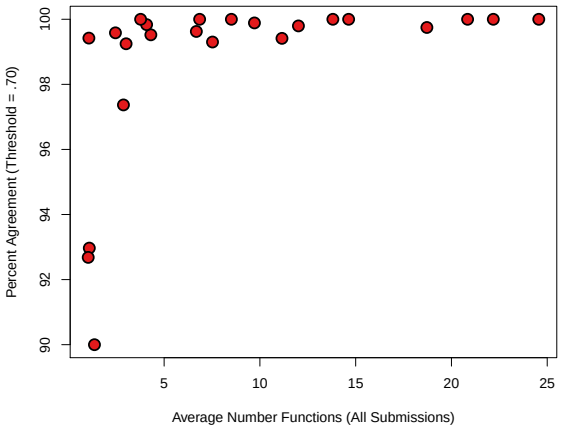
variable would be most accurate is when the participant reports that their submission is based on another participant's submission. Therefore, only these instances of the 'based on' variable are used to determine the quality of the similarity threshold T. Figure 2.2a shows the results of this comparison. As seen in the figure, using this criteria, lower threshold values will always outperform higher threshold values because lower thresholds identify fewer function types, thereby merging the evolutionary trajectory into fewer branches. Therefore, to avoid the collapse to a few evolutionary branches, a higher threshold is preferred, but there should be near perfect agreement with the subset of 'based on' information used. Either a threshold of $T = .70$ (with median agreement = 99.7% and mean = 98.6%) or $T = .85$ (with median agreement = 99.3% and mean = 97.5%) are reasonable and both choices produce similar results. For the remainder of this chapter, a threshold of $T = .70$ is used, but the same results for $T = .85$ is provided in Appendix A.3. This decision was also motivated by exploring the effect of code changes on parse trees, which is included in Appendix A.2. Furthermore, the contests that had the lowest amount of agreement with the 'based on' information had the fewest functions and smallest node counts (see Figures 2.2b and 2.2c), and thus fewer lines of code needed to be changed in order for the function to be classified as a different function. This limitation seems unavoidable unless an approach that used absolute differences in parse trees rather than percent differences, but this would be disadvantageous for larger and more complex functions, which is the more interesting case to study.

Figure 2.3 shows the complete history of the best-so-far submissions for the most recent programming contest, 'Tiles'. In the figure, each small square represents a function from a submission and each column of squares is a complete submission. Each row shows the usage of a function type where the color of the square displays how similar the function is with the originating function of that type. The color coding shows how some function types remain the same throughout their lifespan, but others change drastically.

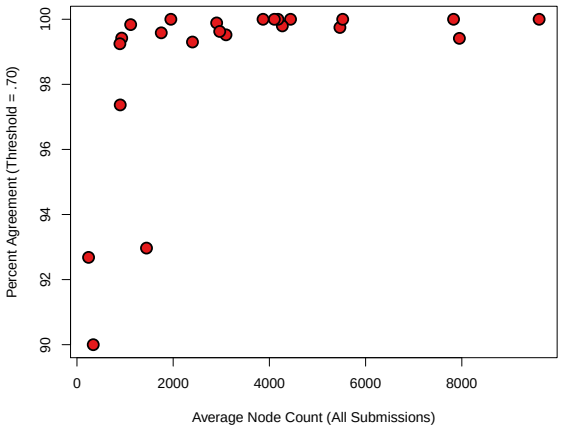
As noted earlier, during the first two days of a contest, the submissions were not visible



(a) Range of Similarity Thresholds



(b) Threshold = .70



(c) Threshold = .70

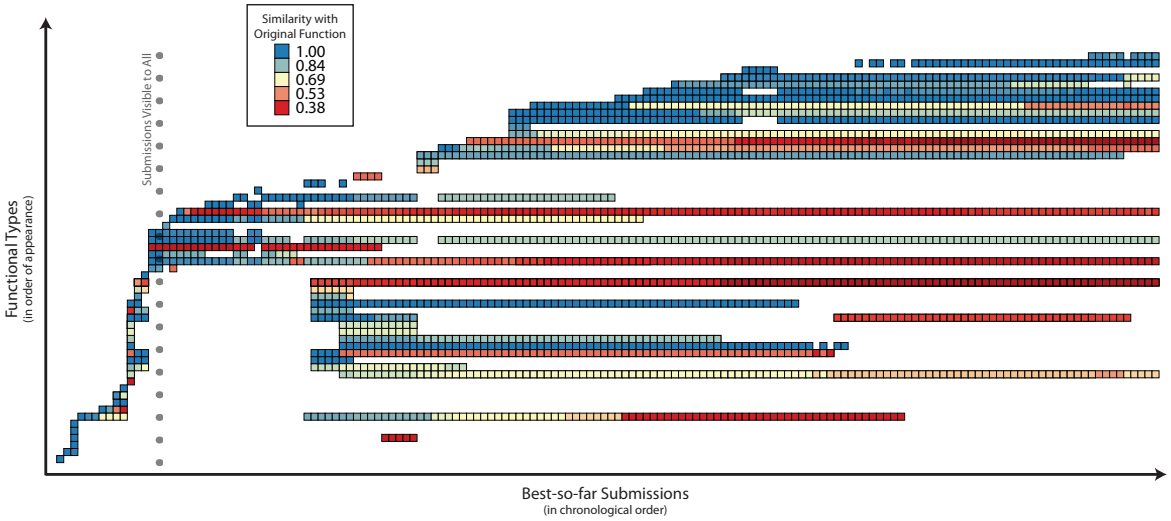
Figure 2.2: This figure shows how the similarity threshold was chosen. a) Various similarity thresholds were compared to see which threshold had the best agreement with the 'based on' variable supplied by the contest participants. The highest threshold that introduces the less disagreement with the participant response is desired. b) While investigating threshold = .70 further, the most disagreement occurs when there are a small number of functions c) and a small node count, which is expected.

to all participants. The dotted line shows when all submissions are viewable by all participants. For this contest, once the submissions are viewable by all participants, there is some additional development on the best-so-far submission, but then the participants start improving upon an earlier idea. Finally, there is a new idea that dominates the remainder of the contest (represented by the cluster of squares at the top of the figure). The figure shows how there is a steep increase in reuse of function types immediately after the submissions are visible to others, which gives a strong indication of collaboration within the contest.

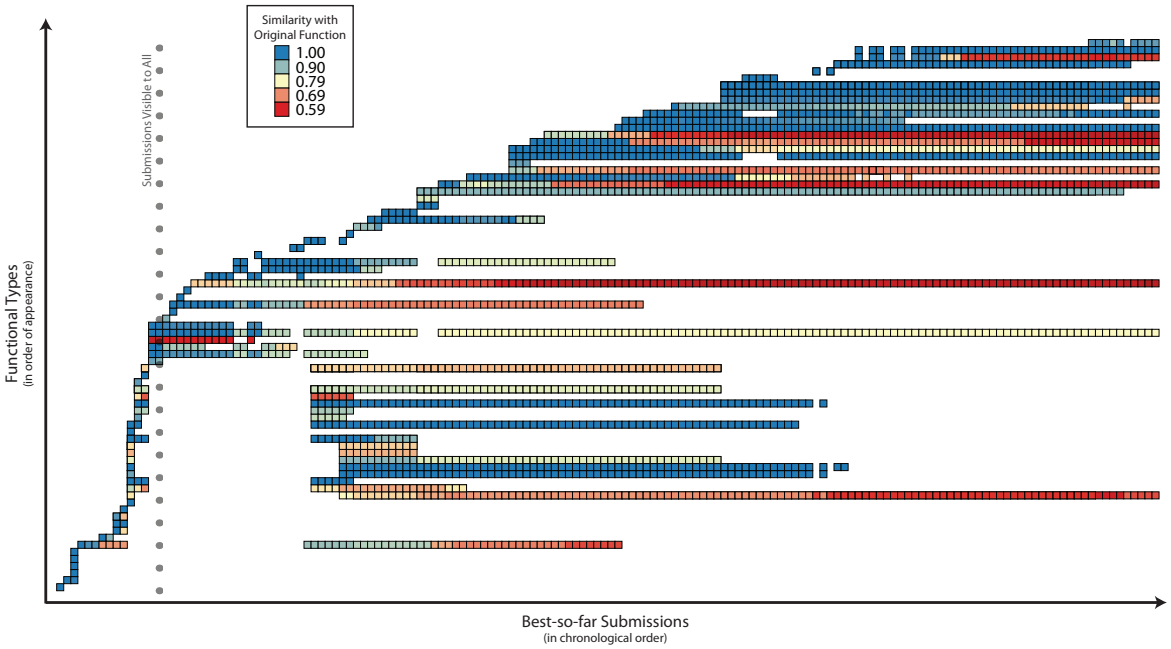
Figure 2.3 provides an intuitive sense of the evolution of the function types used in the contests. Furthermore, it shows that with either threshold, .70 or .85, the contest unfolds in a similar way. This figure also shows how higher similarity thresholds partition functions into more function types, while lower thresholds consolidate functions into fewer types.

2.6.2 Distributions of Projects and Developers

Figure 2.4 shows the developers per project and Figure 2.5 shows the projects per developer. All of these plots are on a log-log scale. Some of the tails of the distributions were not ruled out as being a power law (see Tables 2.5 and 2.6) using the maximum likelihood method described in Clauset et al. [88]. However, I do not want to overstate the fit of the distributions to a power law because of the small sample sizes, although most distributions display a heavy tail, which is a signature of complex systems. Clauset et al. [88] note that generally at least 50 observations are required to get reliable parameter estimates for a power law. This is a problem for some of the distributions used here because only the tail is being fitted to the distribution (refer to n_{tail} values in Tables 2.5 and 2.6). These distributions provide evidence that heavy tail distributions can develop over a relatively short time period (i.e., one week) with a modest number of observations.

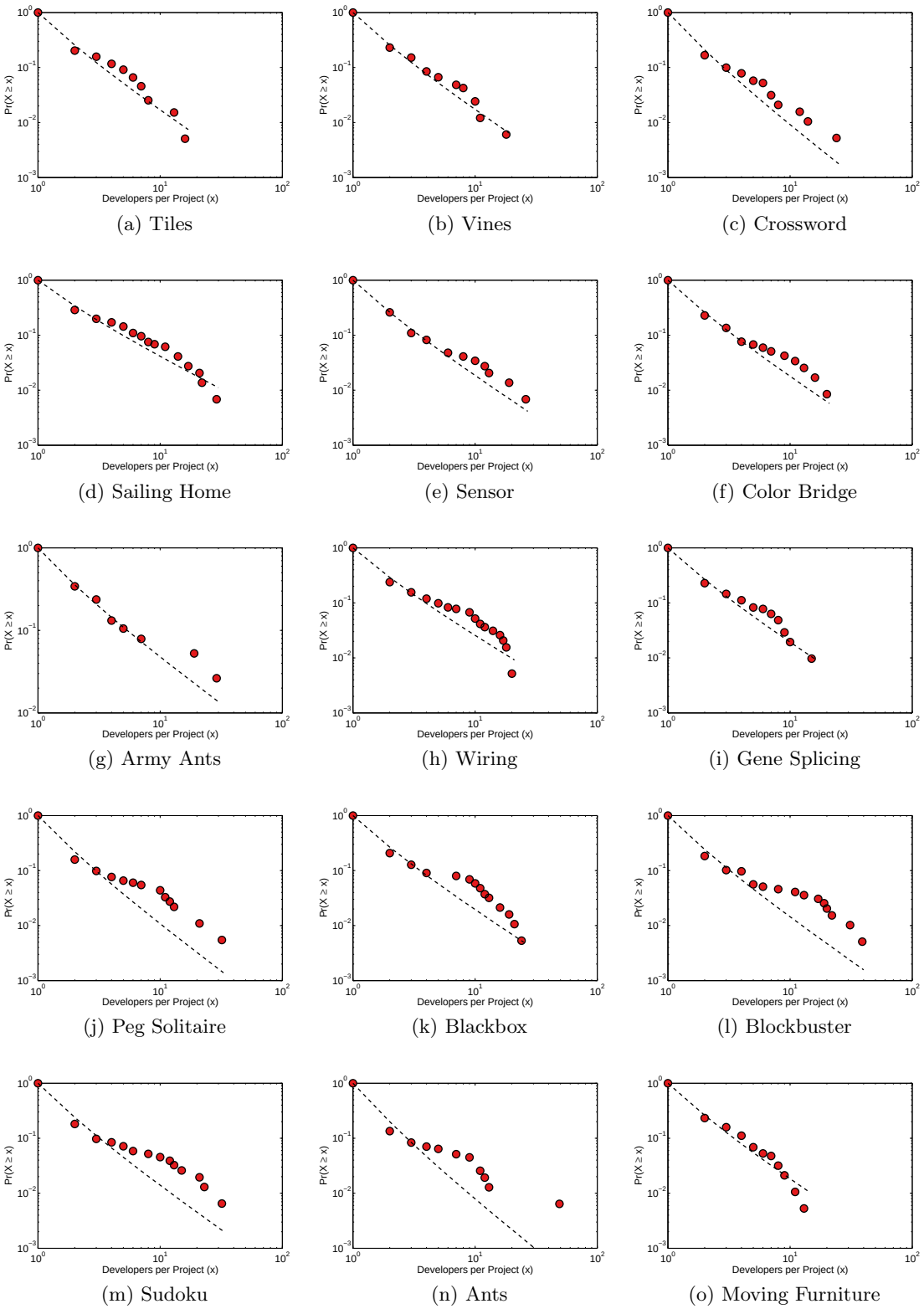


(a) Similarity Threshold = .70



(b) Similarity Threshold = .85

Figure 2.3: The evolution of function types within the most recent programming contest ('Tiles'). For both similarity thresholds, a similar pattern appears. The results discussed in this chapter are not overly sensitive to the choice of threshold. From this figure, you can see the evolution of all of the best-so-far submission for the contest. Each small square represents a function within a submission. A column of squares represent an entire submission. Each row shows how a function type is used over time where the color of the square shows how similar the function is with the originating function of that type. The dotted line shows when all submissions are viewable by all participants.



(a) Tiles (b) Vines (c) Crossword

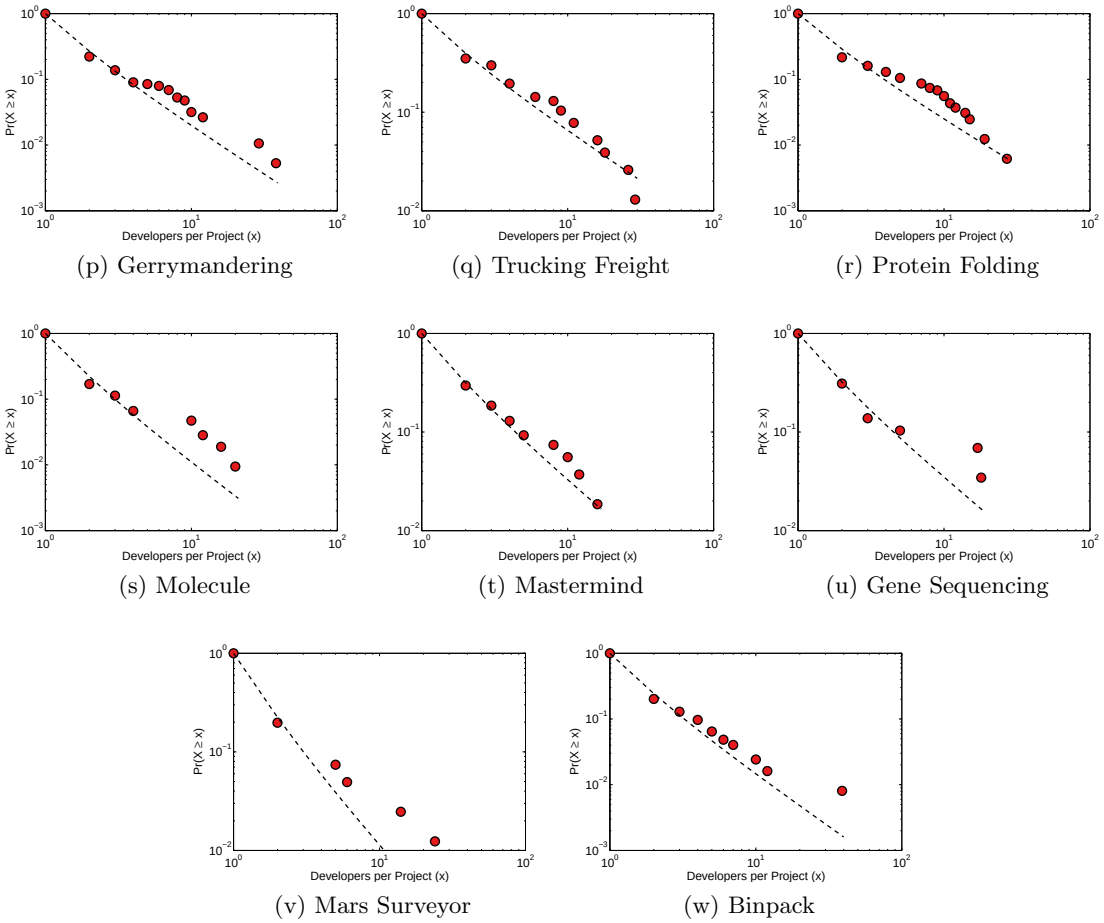
(d) Sailing Home (e) Sensor (f) Color Bridge

(g) Army Ants (h) Wiring (i) Gene Splicing

(j) Peg Solitaire (k) Blackbox (l) Blockbuster

(m) Sudoku (n) Ants (o) Moving Furniture

Figure 2.4: Developers per project for the 23 programming contests ($T = .70$).

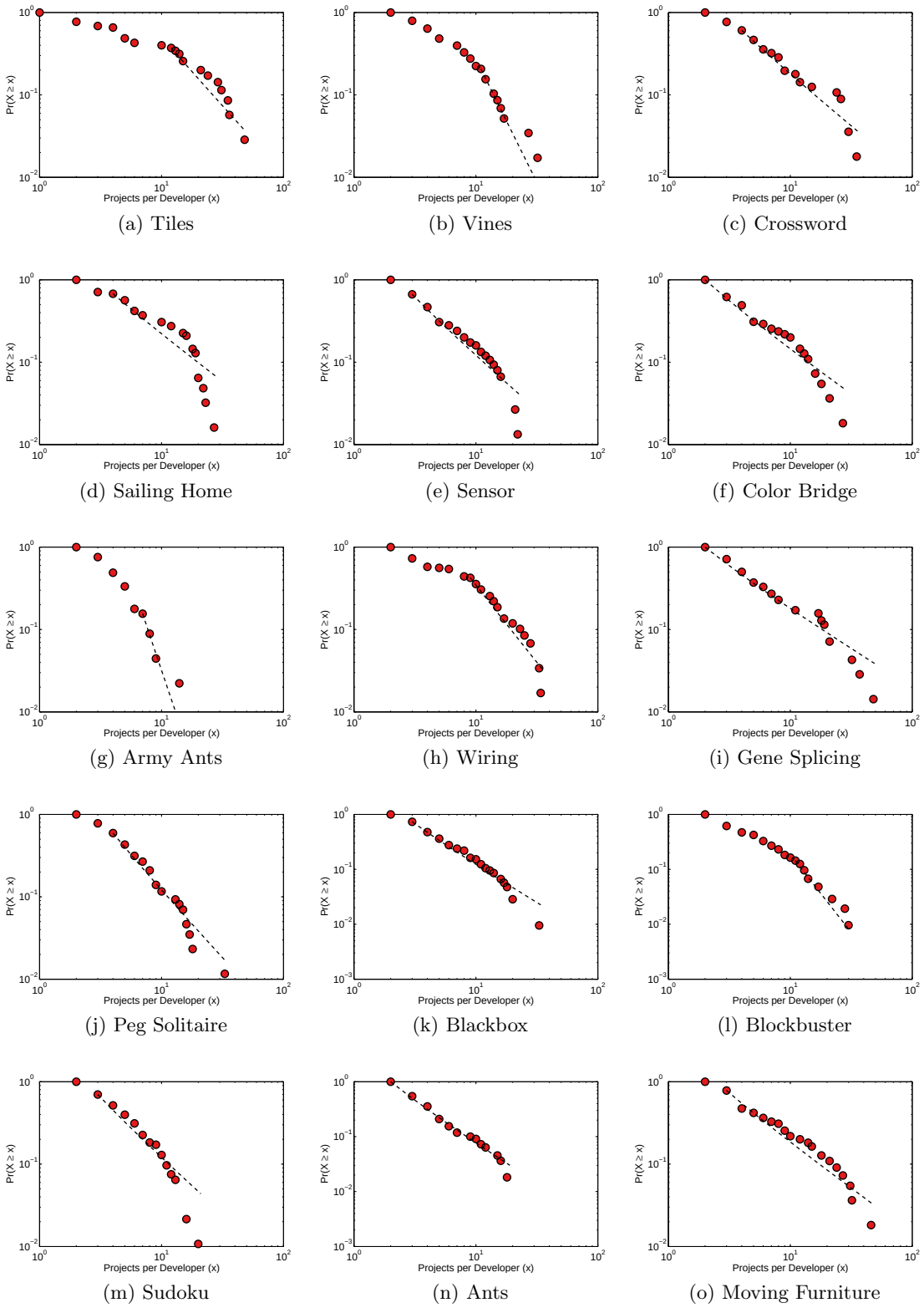


(p) Gerrymandering (q) Trucking Freight (r) Protein Folding

(s) Molecule (t) Mastermind (u) Gene Sequencing

(v) Mars Surveyor (w) Binpack

Figure 2.4: Developers per project for the 23 programming contests ($T = .70$).



(a) Tiles (b) Vines (c) Crossword

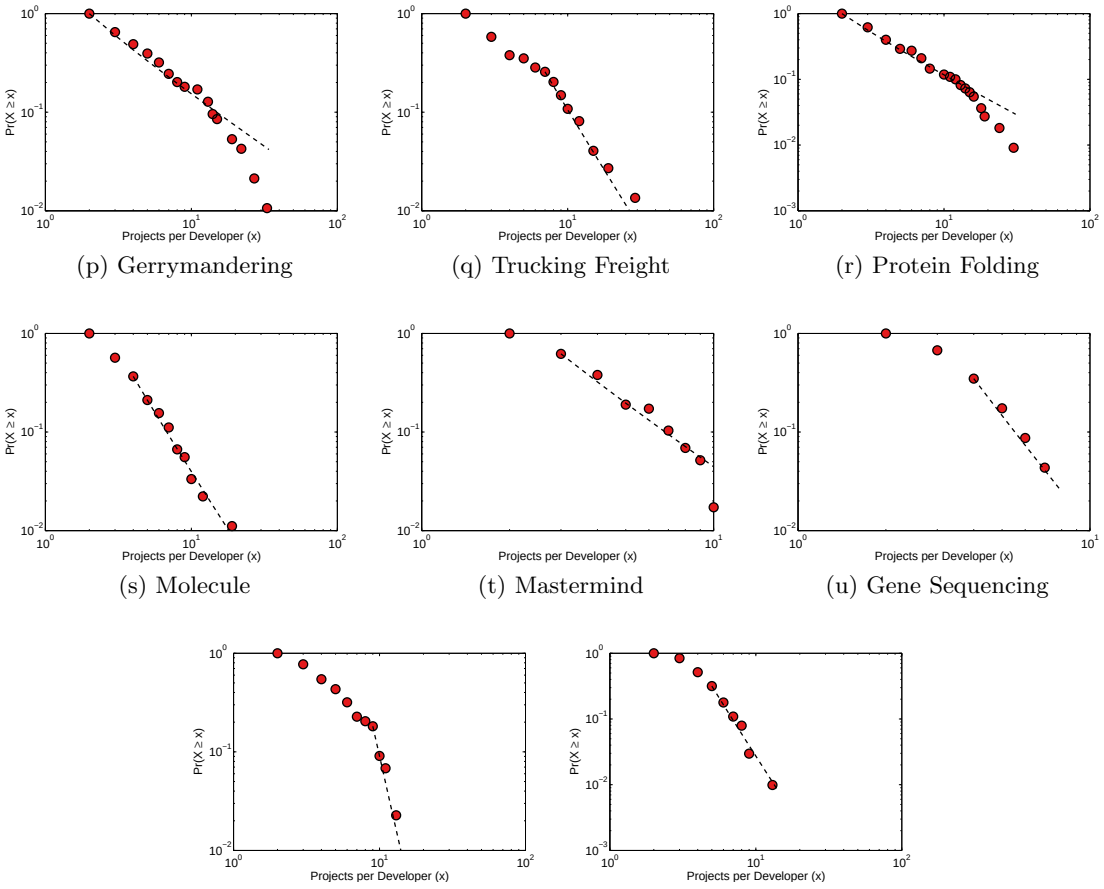
(d) Sailing Home (e) Sensor (f) Color Bridge

(g) Army Ants (h) Wiring (i) Gene Splicing

(j) Peg Solitaire (k) Blackbox (l) Blockbuster

(m) Sudoku (n) Ants (o) Moving Furniture

Figure 2.5: Projects per developer for the 23 programming contests ($T = .70$).



(p) Gerrymandering (q) Trucking Freight (r) Protein Folding

(s) Molecule (t) Mastermind (u) Gene Sequencing

(v) Mars Surveyor (w) Binpack

Figure 2.5: Projects per developer for the 23 programming contests ($T = .70$).

Table 2.5: Parameters for the developers per project distributions ($T = .70$).

Contest Name	n	$\hat{x}_{min}$	$\hat{\alpha}$	n_{tail}	p
Tiles	197	1	2.50	197	0.00
Vines	165	1	2.49	165	**0.21**
Crossword	191	1	2.73	191	0.00
Sailing Home	146	1	2.16	146	0.02
Sensor	146	1	2.47	146	**0.60**
Color Bridge	118	1	2.48	118	**0.25**
Army Ants	38	1	2.11	38	**0.92**
Wiring	193	1	2.34	193	0.00
Gene Splicing	206	1	2.45	206	0.03
Peg Solitaire	183	1	2.67	183	0.00
Blackbox	188	1	2.45	188	0.00
Blockbuster	196	1	2.56	196	0.00
Sudoku	154	1	2.57	154	0.00
Ants	156	1	2.79	156	0.00
Moving Furniture	189	1	2.47	189	0.08
Gerrymandering	189	1	2.44	189	0.01
Trucking Freight	77	1	1.99	77	**0.17**
Protein Folding	162	1	2.35	162	0.00
Molecule	106	1	2.66	106	0.01
Mastermind	54	1	2.25	54	**0.75**
Gene Sequencing	29	1	2.23	29	**0.53**
Mars Surveyor	81	1	2.65	81	**0.19**
Binpack	124	1	2.56	124	0.04

bold values are statistically significant ($p > .10$)

Table 2.6: Parameters for the projects per developer distributions ($T = .70$).

Contest Name	n	$\hat{x}_{min}$	$\hat{\alpha}$	n_{tail}	p
Tiles	35	12	2.62	13	**0.26**
Vines	58	11	3.92	12	**0.52**
Crossword	56	4	2.23	34	**0.41**
Sailing Home	62	4	2.13	42	0.00
Sensor	75	3	2.29	50	**0.36**
Color Bridge	55	2	2.06	55	**0.23**
Army Ants	45	7	5.20	7	**0.44**
Wiring	59	9	2.82	25	**0.38**
Gene Splicing	70	2	1.94	70	0.06
Peg Solitaire	86	4	2.60	51	**0.21**
Blackbox	105	3	2.35	77	**0.15**
Blockbuster	104	11	3.77	15	**0.34**
Sudoku	93	3	2.33	65	0.00
Ants	110	2	2.42	110	**0.30**
Moving Furniture	55	3	2.09	43	**0.13**
Gerrymandering	94	2	2.03	94	**0.15**
Trucking Freight	74	7	3.35	19	**0.60**
Protein Folding	110	2	2.19	110	**0.12**
Molecule	90	4	3.25	33	**0.49**
Mastermind	58	3	3.00	36	0.08
Gene Sequencing	46	4	4.51	16	**0.38**
Mars Surveyor	44	3	2.44	8	0.03
Binpack	101	5	4.31	32	**0.40**

bold values are statistically significant ($p > .10$)

2.7 Analysis of Problem Solutions

The analysis of problem solutions is centered around what Schumpeter [33] refers to as creative destruction. Schumpeter's theory of creative destruction describes how existing technologies can be replaced by a new technology [33]. The programming contest data provides a microcosm to study the "perennial gale of creative destruction" [33, pg. 84].

2.7.1 Evidence of Creative Destruction

New approaches to the problem are introduced throughout the contest. If creative destruction is present in the data, then it would be expected that when a new, better submission is introduced, then it would be adopted by participants and replace some of the previously used approaches. Therefore, if you look at the submission diversity before and after the introduction of an improvement, then it would be expected that the variety of approaches being used would decrease (Algorithm 2.2 explains how the change in diversity was calculated). The submission diversity was calculated by looking at the 10 submissions by unique authors before and after each submission and counting the number of unique approaches. An approach is defined as a unique combination of function types as defined earlier in this chapter. A negative change in submission diversity means that the submission population became more homogeneous and a positive change in diversity means the population became more heterogeneous.

Algorithm 2.2 Change in Solution Diversity

for all $submission_i$ **do**
  $before_i \leftarrow count\_approaches\_before(submission_i)$ {by 10 unique authors}
  $after_i \leftarrow count\_approaches\_after(submission_i)$ {by 10 unique authors}
  $changeInDiversity_i \leftarrow after_i - before_i$
end for

From Figure 2.6, many of the best-so-far solutions coincide with periods of reduced submission diversity. In the figure, the horizontal axis displays the submissions in chronological order for the Color Bridge contest. The vertical axis displays the change in diversity as calculated by Algorithm 2.2. The points on the figure indicate the best-so-far submissions with downward pointing triangles displaying creative destruction. However, not all best-so-far submissions caused creative destruction and those points are marked with upward pointing triangles. Obviously, there are other factors at work and every best-so-far submission isn't going to reduce the submission diversity.

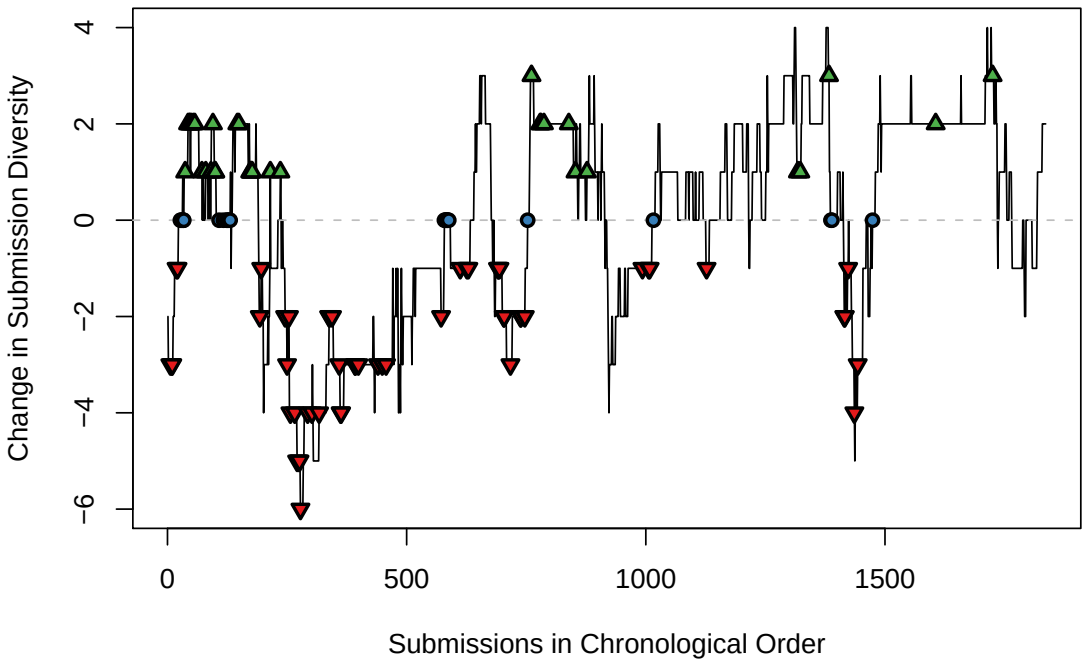


Figure 2.6: Creative destruction in the Color Bridge contest. The points indicate the locations of the best-so-far submissions with red downward pointing triangles showing creative destruction. Green upward pointing triangles reveal best-so-far submissions that increased the submission diversity. Blue circles indicate best-so-far submissions that had no change in submission diversity.

Figure 2.7 shows the difference in the average change in diversity for the best-so-far solutions versus all other submissions (i.e., the submissions that were not the best-so-far). The majority of the contests (17 of 23) show a greater decrease in diversity for the best-so-far submissions. However, only six of the contests were statistically significant with one of those in the wrong direction. Statistical significance was determined using a permutation test [89]. A permutation test is a non-parametric, resampling method were the observations from the two conditions are randomly sampled to determine significance.[7] The Kolmogorov-Smirnov test is another common non-parametric test, but was not used because it requires continuous values and the difference in diversity can only be discrete, whole numbers. Also, it is important to note there is a tendency of the submission diversity to decrease over the duration of a contest. This can be seen in Figure 2.7 by observing the tendency of the diversity values to be below zero. As the contest progresses and better approaches are discovered, less effort is placed on exploration of new techniques and more effort is placed on the exploitation of the best solutions.

2.7.2 Reuse of Improvements

Since results were modest when examining the change in diversity before and after the best-so-far submissions, another analysis was performed to look for more convincing evidence of creative destruction. Instead of looking at the reduction in diversity, this analysis looks for the adoption of a new innovation by other participants. Using this view, when an improvement is introduced, it is expected that it will be reused by others more than submissions that do not make an improvement over the best-so-far. For this analysis, the number of times that the approach was used by 10 unique authors (not including the original author) was counted for each submission (see Algorithm 2.3). As seen in Figure 2.8, the number of times the best-so-far submissions are reused compared to the other submissions is statistically significant in 19 of the 23 contests. Furthermore, there is high levels of reuse for all contests and all submissions giving a strong indicator that the contests are facilitating

[7]For the permutation test, I used the software that was included with Maindonald and Braun's book [90].

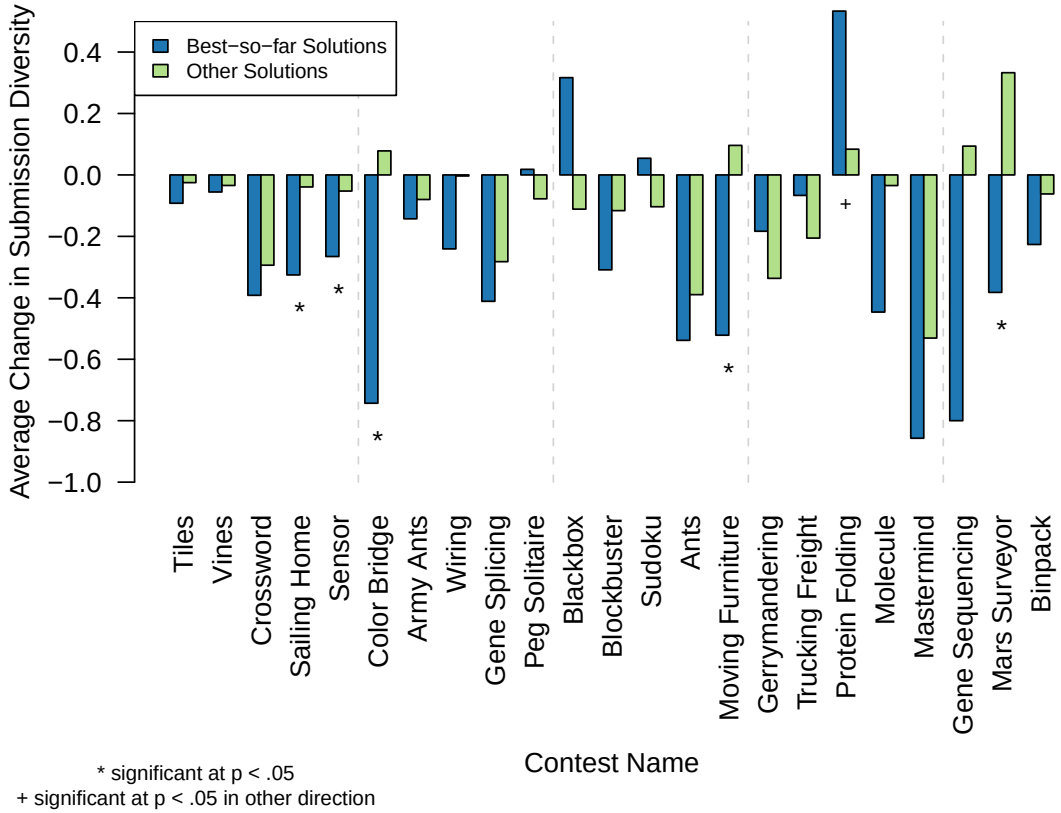


Figure 2.7: Creative destruction in the programming contests using the change in submission diversity (threshold $T = .70$). Most contests (17 of 23) have a larger drop in submission diversity with best-so-far solutions compared to all other submissions. Also, note that generally submission diversity reduces during the cause of the contests (represented by the majority of negative values).

collaboration. This also agrees with the result from Gulley and Lakhani [83], which found that solutions with higher ranks at time of entry had more lines of code reused by other participants. Moreover, the analysis method used by Gulley and Lakhani [83], quasi-maximum likelihood fixed effects poisson regression, was quite different than the approach used here, thereby providing support that this is a robust finding.

Algorithm 2.3 Number of Solution Reuses

for all $submission_i$ **do**
  $numReuses_i \leftarrow count\_reuses(submission_i)$ {by 10 unique authors}
end for

The results from the previous two analyses, the creative destruction analysis and the reuse of improvements analysis, seem to be conflicting with one another. However, when analyzing the change in submission diversity, the analysis did not take into account that the best-so-far submission could be adding to the diversity. In other words, the approach used in the best-so-far submission is more likely to be in the population after it was the best, then before, thereby adding to the submission diversity. Also, the best-so-far submission could be modified beyond the similarity threshold and again, add to the submission diversity. The results reporting the number of reuses overcome these shortfalls and provide more evidence of creative destruction in the programming contests. Furthermore, since solutions are frequently reused by different authors, this demonstrates that the amount of collaboration between authors is quite high.

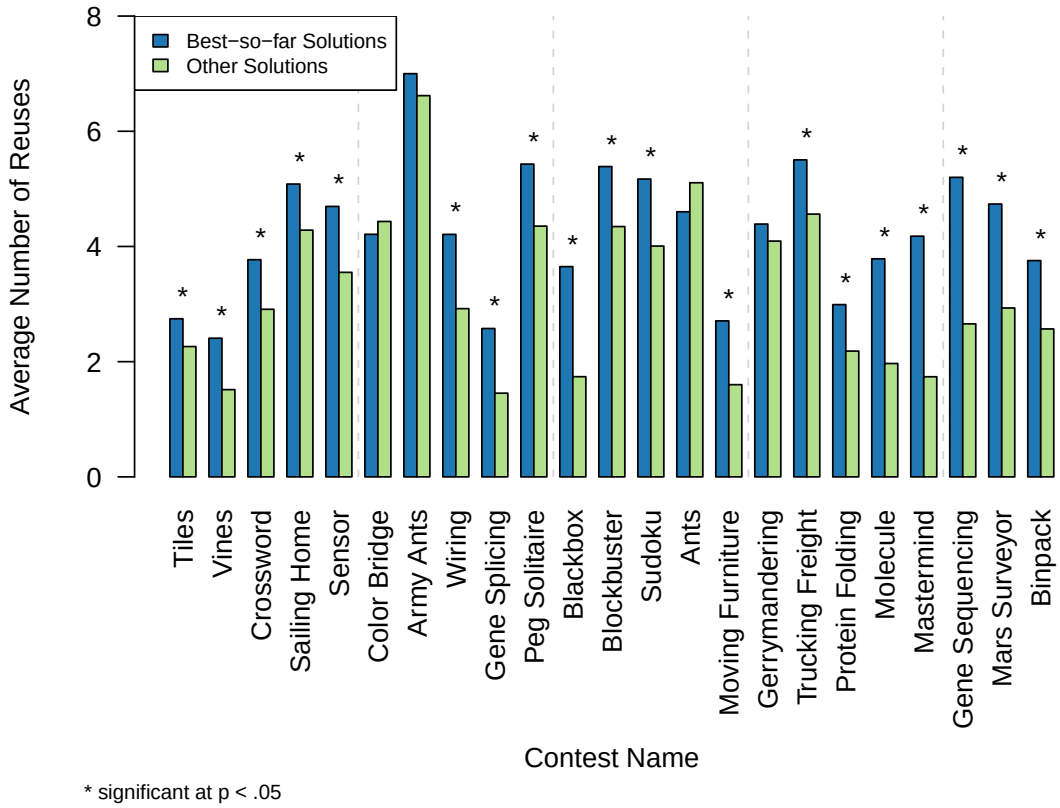


Figure 2.8: Submission reuse in the 23 programming contests (threshold $T = .70$). Most contests (19 of 23) have a statistically significant difference in the amount of reuse between best-so-far submissions and other submissions.

2.8 Discussion

Much of the analysis in this chapter required careful examination of the functions (the building blocks), which is computationally expensive, but the underlying concepts under study are quite simple. The programming contest data provides a look inside the innovation process at a low level where a bottom up view of the process can be obtained. Furthermore, these analyses provide a quantitative way of exploring the problem being solved, the problem-solvers, and the problem solutions. With a quantitative analysis, the importance of various factors can be systemically compared to determine which are the best predictors of innovation.

The analysis of problem characteristics found that problem modularity and size were significant predictors of innovation. Moreover, problem modularity had an effect size (measured by R-squared values) approximately double that of problem size. This provides quantitative evidence that problem modularity is a better predictor of innovation than problem size. Problem complexity was not a statistically significant predictor of innovation even in the contests that did not include complexity as part of the scoring criteria (see Appendix A.1). This type of empirical data analysis allows for comparison between difference aspects of the problem-solving task.

When studying the problem-solvers, the analysis of the distributions of developers per project and projects per developer show that heavy tailed distributions can emerge in a short period of time. It also shows that the MATLAB programming contests have similar patterns as what is seen in the open source software community. Therefore, it may be a more controlled way of studying open source software and its evolution.

Finally, the analysis of problem submissions shows some evidence of a drop in submission diversity after a best-so-far solution, which provides limited evidence of creative destruction. These results were not totally convincing so the analysis of number of reuses of best-so-far submissions provided a clearer picture where new innovations were more likely to be reused by other participants. This provides evidence that a new improvement overtakes existing solutions, but since there is little drop in submission diversity, there seems to be a

simultaneous spurring of more innovative solutions. Therefore, the creative destruction from a new innovation coincides with a spawning of new ideas. There appears to be two forces at work when a new innovation is introduced. The first is Schumpeter's 'process of creative destruction' and the second is what I refer to as the 'process of imitative construction'. While the process of creative destruction causes old products to be eliminated, the process of imitative construction promotes new products due to the introduction of a new idea.

The analyses discussed in this chapter all deal with static fitness landscapes. These landscapes are appropriate for innovation that occurs over a relatively short time period. By allowing the landscape to remain static, it is much simpler to model. In the next chapter, I model groups of collaborating agents working to solve problems with static fitness landscapes.

Chapter 3: Models of Static Fitness Landscapes

In this chapter, the results from the data analysis in the previous chapter are used in the creation of computational models. The computational models extend the types of quantitative analysis that are possible. Evolutionary computation (i.e., genetic algorithms) and the NK model are used as starting points for the computational models. The computational models show how simple mechanisms can reproduce patterns from the MATLAB programming contests without sophisticated modeling of agents' cognitive abilities. In this chapter, just like the previous chapter, the quality of a product does not change during its lifetime, thereby creating a static fitness landscape.

The chapter begins with an overview of the background concepts used in the remainder of the chapter followed by the model description and model results. Finally in this chapter, the results from the computational models are compared with mathematical models that were generated using only empirical data. This allows for further validation of the computational models and the development of a better mathematical model of innovation.

3.1 Research Questions

Question 1: Which of the following computational models most closely resemble the empirical data from the programming contests?

[A:] Artificial problem-solving agents using the Monte Carlo method to randomly generate new solutions with no consideration of previously generated solutions

[B:] Artificial problem-solving agents that are locally optimizing problem solutions without collaboration

[C:] Artificial problem-solving agents that are using selection and random recombination of others' solutions to search for better solutions

Question 2: Once a computational model is created, can it provide a clearer picture of the innovation process by permitting a systemic sweep of model parameters?

Question 3: Can the computational models be integrated with mathematical models that were generated using only empirical data to help develop stronger support for both types of models?

The first goal is to compare the results from the computational models with the empirical data gathered from the MATLAB programming contests (Question 1). The computational model using the Monte Carlo method provides a baseline of comparison for the other two computational models. Using computational models from the first question, a systemic sweep of problem parameters, size and modularity, was performed to determine their effect on innovation (Question 2). The last task is to integrate what was learned from the computational model back into the mathematical model to further strengthen and develop the results from both types of models (Question 3). The first question is explored in the next section, Section 3.2, and the second question is addressed in Section 3.3. Finally, the last question is addressed in Section 3.4.

3.2 Modeling the Programming Contests

The programming contests have many variables that could be confounding the results. For example, in each contests, there are a different number of participants. This requires that the innovation measures be normalized by the number of participants (e.g., innovations per participant and percent innovators). Furthermore, there is no systemic exploration of problem size and modularity within the programming contests, thereby some parts of the state space are more sparse than others. This can be overcome with computational models by performing a parameter sweep of the problem size and modularity variables. Before discussing the parameter sweep results, the empirical data is used to determine which model most closely replicates the results from the programming contests. The idiosyncratic characteristics from each of the programming contests are used to parametrize the fitness

landscape used in the computational models. Then, the model results are compared to the empirical measures of innovation to see which model gives the most accurate results. The following three sections describe the computational models used in this analysis. These models are modeling a group of problem-solvers that are concurrently working on a single problem, but each model uses a distinctive mechanism to do this.

3.2.1 Monte Carlo Model

The Monte Carlo model is the simplest of the three computational models. Artificial agents generate solutions to the problem over multiple generations, in this case, for a total of 1,000 generations. Each agent creates a single solution during each generation and every solution is randomly generated without any consideration of previously generated solutions. The best-so-far solution is recorded for each generation during the execution of the model. As in all of the computational models, the problem being solved is defined by a NK landscape, which is discussed in detail later in this chapter. Unlike the other two computational models, the agents in this model do not use previously generated problem solutions to aide in the creation of new solutions. Therefore, these agents do not use previously gained knowledge during the innovation process, but are given the same number of attempts at solving the problem. This provides a baseline for comparison with the other two computational models.

3.2.2 Parallel NK Model

Kauffman's NK model [49] is the starting point for this model and the landscape for the NK model is used for all of the computational models in this chapter. The original NK model uses a simple mechanism for modeling a single individual that is searching for better solutions on a fitness landscape. To move to a group problem-solving process, I simply have multiple problem-solvers concurrently working on the same problem. In the NK model, the landscape specifies how the elements of a solution (or individual) influence its fitness. The interactions between the solution elements and fitness components are called epistatic relationships [49]. The fitness landscape represents the problem that is being solved by the

problem-solvers. The N parameter is the size of the fitness landscape, thereby represents the size of the problem being solved. Each problem solution generated is N 'genes' long where a gene can have a binary value (0 or 1). In the standard NK model, The i^{th} gene always influences the i^{th} fitness component, which is also followed here. The K parameter is the number of other fitness components that each of the solution elements (genes) influence. Therefore, as K increases, the number of gene to fitness component interactions, or epistasis, increases.

The fitness of an individual (solution) is described by the following formula:

$$F = \frac{1}{N} \sum_{i=1}^{N} f_i \tag{3.1}$$

The f_i variables are the values for each of the fitness components and the overall fitness of the individual (F) is the average of those fitness component values. The fitness component values (f_i values) are randomly drawn from a uniform distribution from 0.0 to 1.0. For each unique combination of the $K+1$ genes that influence a fitness component, there is a random number generated from the uniform distribution [49]. Therefore, a fitness component could have 2^{K+1} different possible values. Even when only generating fitness component values as they are needed, the NK landscape can be very large. This makes some versions of the NK model NP-complete [91] which will be discussed more later.

The K parameter provides a systemic way to tune the amount of ruggedness in the landscape. For example, when $K = 0$, each element only influences one fitness value, so for each element the optimal individual can be found using a hill-climbing algorithm. When $K = N - 1$, all elements influence all fitness values, thereby if a single element changes then every fitness component values changes. Therefore, creating a totally uncorrelated fitness landscape where there is no relationship between a location on the fitness landscape and neighboring locations [55].

In Kauffman's most basic formulation of the NK model, the agents (problem-solvers)

searching over the NK landscape use a steepest-ascent hill-climbing algorithm. The agent starts on a random location of the landscape. Then the agent evaluates each of the neighboring locations on the landscape and moves to the best location. The neighboring locations are the locations on the landscape that are different from the current location by one gene, thereby there are N neighboring locations. The agent continues until it cannot find a neighboring location that has a better fitness, or in other words, the agent stops when it reaches a local optimum [49].

The parallel version of the NK model included here simply includes multiple agents concurrently hill-climbing over the landscape. The agents are goal-directed, but are working independently. Later in this chapter, another computational model is discussed that uses a genetic algorithm. In the genetic algorithm model, agents randomly recombine other agents' solutions, which enables collaboration in the model. Before discussing the genetic algorithm model, the three types of NK landscapes are introduced that are used by the computational models.

Types of NK Landscapes

There are numerous ways that the K epistatic relationships can be distributed. In Kauffman's original NK model, there were two arrangements of epistatic relationships. Figure 3.1a and 3.1b show examples of Kauffman's original landscapes. In the adjacent landscape, the i^{th} gene and its K neighboring genes affect the i^{th} fitness component. In the random landscape, the i^{th} gene and K other randomly chosen genes affect the i^{th} fitness component [49]. It is interesting to note that the adjacent landscape can be optimized in polynomial time (if $K \in O(logN)$ [92]), but the random landscape has been found to be NP-complete [91]. Moreover, Kauffman surprisingly found quite similar results using both landscapes [49], which is also the case for the results presented here.

Altenberg's generalized NK model [52] was used to introduce modularity into the third landscape (see Figure 3.1c). In the generalized NK model, any arrangement of epistatic relationships between genes and fitness components can be specified and the K value is not

necessarily the same for all fitness components [50] [51] [52]. In the modular landscape, the N genes were partitioned into M modules.

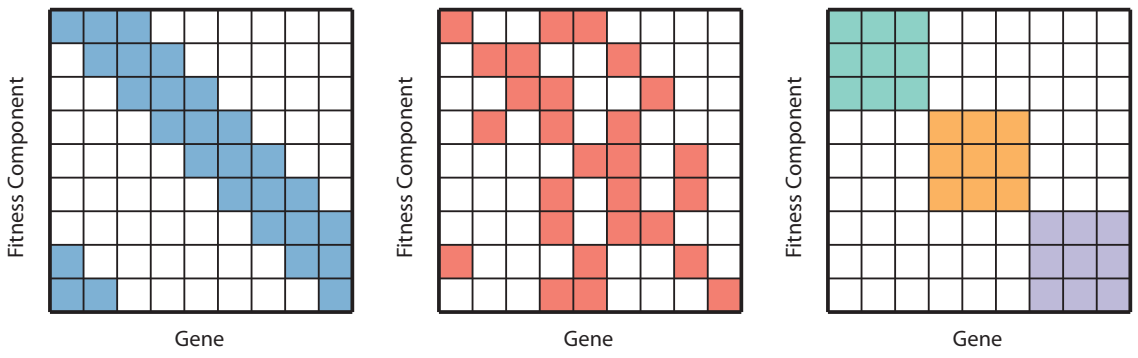
$$M = \frac{N}{K+1} \tag{3.2}$$

Therefore, each fitness component had approximately $K+1$ epistatic relationships, just like the other fitness landscapes.

$$K = \frac{N}{M} - 1 \tag{3.3}$$

In some of the landscapes, the K value was not a whole number, so some fitness components had more epistatic relationships than others, but this minor discrepancy should have little effect on the results.

The three fitness landscapes allowed the epistasis to remain nearly constant while changing the arrangement of epistatic relationships. The modular landscape had the additional benefit of being able to systemically adjust the amount of modularity in the landscape during a parameter sweep.



(a) Adjacent (b) Random (c) Modular

Figure 3.1: The different types of NK landscapes used for the computational models. The first two, adjacent and random, are the same as Kauffman used [49]. The modular landscape divides the landscape into modules. Also, notice that the epistasis of each of the landscapes is the same.

The parallel version of the NK model simply has multiple agents searching over the landscape using hill-climbing. The agents are goal-directed, but are working independently. In the next computational model, discussed in the next section, the agents are randomly recombining other agents' solutions, thereby allowing for collaboration between agents in the model.

3.2.3 Genetic Algorithm Model

Genetic algorithms are primarily used for solving problems, but have also been applied to the study of evolutionary processes [93]. Here a genetic algorithm is used to determine how closely the behavior of the genetic algorithm matches the behavior of human problem-solvers. I am not proposing that humans solve problems in the same way as a genetic algorithm; I am simply proposing that the patterns found in human problem-solvers may also be found in the results of a genetic algorithm. Furthermore, the NK model has been suggested as a systemic way of studying the effect of epistasis during the execution of a genetic algorithm [94]. For this model, a small variation was made to the genetic algorithm where the population grows during the execution of the model. This makes the model more similar to the programming contest data where the number of submissions grew during the course of the contest. Just as in the standard genetic algorithm, the genetic algorithm discussed here still depends on the process of selection, inheritance, and variation to provide the mechanism for the problem solving process.

In genetic algorithms, each individual represents a solution to the problem being solved and consists of a 'genome' that is made up of 'genes'. Each gene could be a real number, but is commonly just a binary value, which is the case for the NK landscapes. The initial population is generally randomly generated. For each successive generation, natural selection chooses the parents for the next generation. Normally each new individual inherits part of its genome from each parent which creates a new combination of existing parts. Just as in the Monte Carlo model, the genetic algorithm was executed for 1,000 generations.

The genetic algorithm used here uses only the crossover operator [71] [95]. The crossover

operator selects portions of each parents' genome to reuse to create a new child. For example in one point crossover, the genes from the first individual are used up to the crossover point and then the genes from the second individual are used. This idea can be extended to multiple crossover points, but one point crossover was used for these model results. The mutation operator is another common operator for introducing variation into the population of a genetic algorithm. Mutation only requires one individual and randomly selects a gene to change. The model results did not depend on mutation being present, so for simplicity, only crossover was used in the results discussed here.

Building Block Hypothesis

The building block hypothesis provides an explanation of how a genetic algorithm solves problems [71] [95]. In this hypothesis, as the genetic algorithm evolves, patterns of genes, or schemata as named by John Holland [71], of high quality will appear in the population. Therefore, Holland's schema theorem [71] states that in the next generation the number of instances that have a schema, h, will have a lower bound expressed by:

$$\left(\frac{\hat{f}_h}{\hat{f}}\right)(1-e_h)\,P_h \tag{3.4}$$

The $\hat{f}_h$ and $\hat{f}$ terms are the average fitness for the individuals with schema h and all individuals in the population, respectively. The P_h value is the proportion of individuals that have schema h. The e_h term is an error term that is determined by the length of the schema and the reproduction operators used. It measures the number of instances of the schema that are destroyed during the reproduction step. In short, the schema theorem states that on average the number of instances with a schema with higher than average fitness will grow exponentially in subsequent generations [71] [72] [96] . The building block hypothesis states that crossover can use these schemata as building blocks to create better solutions of a higher order. For the schemata (the building blocks) to not be destroyed over

generations, it is best for them to be short and have high fitness [71] [95].

Model Parameters

Individuals in the genetic algorithm were selected to reproduce using tournament selection. In tournament selection, a subset of individuals are selected as potential parents and the best individual is used. Tournament selection allows the selection pressure in the evolutionary process to be systemically adjusted. The number of individuals selected for the tournament was varied to study the effect of selection pressure on the evolutionary process. Three sizes were used for tournament selection: 2, 7, and 12. The middle value, 7, is discussed in this chapter and the other two levels of selection are included in Appendix B. The various levels of selection pressure were used to test the robustness of the model results and all three levels of selection pressure produced similar results, thereby providing evidence that the model is not overly sensitive to selection pressure.

The model used standard genetic algorithm approaches as much as possible, but the empirical data motivated one change from the standard genetic algorithm. In the programming contests, the population of solutions grew during the contest. However, generally in a genetic algorithm the population size remains static from one generation to the next as the previous generation disappears once the population for the next generation is created. Therefore, in the genetic algorithm model discussed here, the solutions from previous generations remain in the population for the duration of the model execution. This means that the solution population grows during model execution just as in the programming contests.

3.2.4 Analysis Description

The parameters of the fitness landscape (the NK landscape) were set as closely as possible to the programming contests. Table 3.1 contains the information used to configure the NK landscapes. The model configuration used exactly the same number of agents as the number of participants from the programming contests. The average number of nodes in the parse tree of the contest submissions was used to determine the N parameter of the

Table 3.1: Problem characteristics from the MATLAB programming contests

Contest Name	Number of Participants	Average Node Count	Average Function Count	Number of Improvements
Tiles	35	4439	24.6	141
Vines	57	7828	20.8	108
Crossword	55	4269	12.0	74
Sailing Home	62	2401	7.5	216
Sensor	70	3098	4.3	256
Color Bridge	54	1952	6.9	109
Army Ants	43	930	1.1	7
Wiring	59	5466	18.7	162
Gene Splicing	67	4178	14.6	158
Peg Solitaire	86	2904	9.7	165
Blackbox	105	2967	6.7	61
Blockbuster	102	1116	4.1	191
Sudoku	90	4103	3.8	129
Ants	107	1752	2.5	78
Moving Furniture	54	9607	22.2	164
Gerrymandering	92	5524	13.8	181
Trucking Freight	71	3868	8.5	165
Protein Folding	109	7949	11.2	105
Molecule	89	899	2.9	58
Mastermind	55	892	3.0	29
Gene Sequencing	41	338	1.4	12
Mars Surveyor	44	1446	1.1	79
Binpack	101	241	1.0	61

landscape. However, the node count values were too large to be computationally feasible for the NK landscape. Therefore, to keep the landscape memory space requirement in a reasonable range, the N parameter was set to the average node count divided by 20. The K parameter was calculated using Equation 3.3 with the N parameter set to the node count divided by 20 and the M parameter set to the average number of functions. Therefore, the NK landscapes were parameterized to resemble the programming contests as closely as possible.

The three types of artificial problem-solvers (i.e., Monte Carlo, parallel NK, and genetic algorithm) searched over the landscapes and given enough iterations to reach a steady state. In the parallel NK model, all of the artificial problem-solvers would reach a local optimum

and stop. In the Monte Carlo and genetic algorithm models, one thousand generations were sufficient for the fitness improvements to plateau. There are 23 parameter settings, one for each programming contest, and each setting was executed 20 times.

3.2.5 Analysis Results

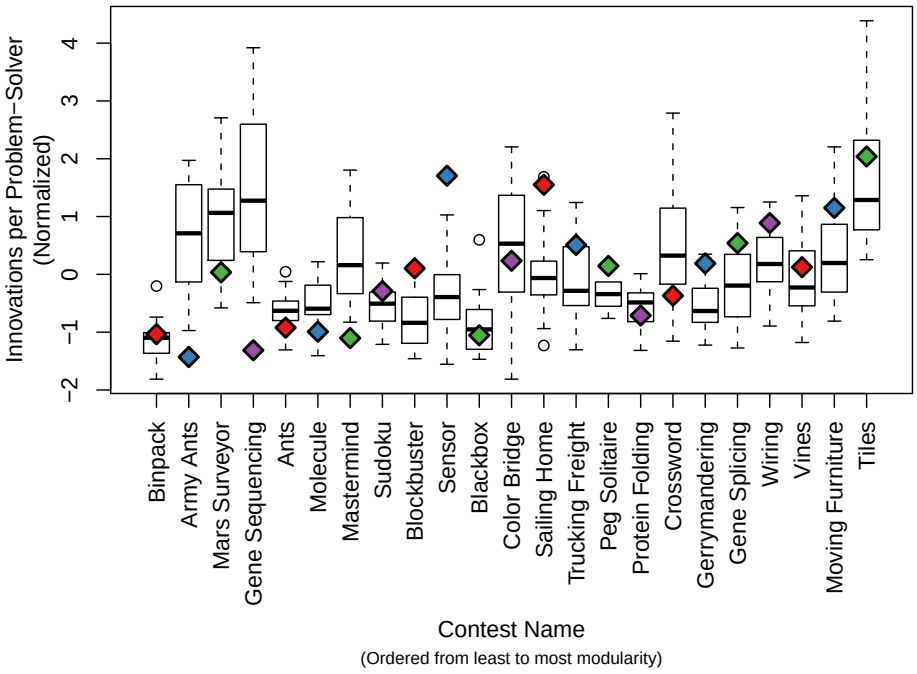
The results are from the three computational models, the Monte Carlo, parallel NK and genetic algorithm models, searching across three different landscapes, adjacent, random, and modular. Moreover, the genetic algorithm model tests the effect of selection pressure by using three sizes of tournament selection: 2, 7, and 12. Therefore, there are fifteen different sets of model results that use the NK landscapes parametrized according to the MATLAB programming contests. Not all of the results are included here for brevity, but the results are included in Appendix B for those interested.

The results of the three models are shown in Figures 3.2, 3.3, and 3.4. In each of the figures, the colored diamonds indicate the empirical results from the programming contest. The color of the diamond does not mean anything explicitly, but allows for easier comparison across plots. For the modular landscape, it was possible to plot the results on a scatterplot as well. This shows how the variation in modularity of the various contests are not evenly distributed.

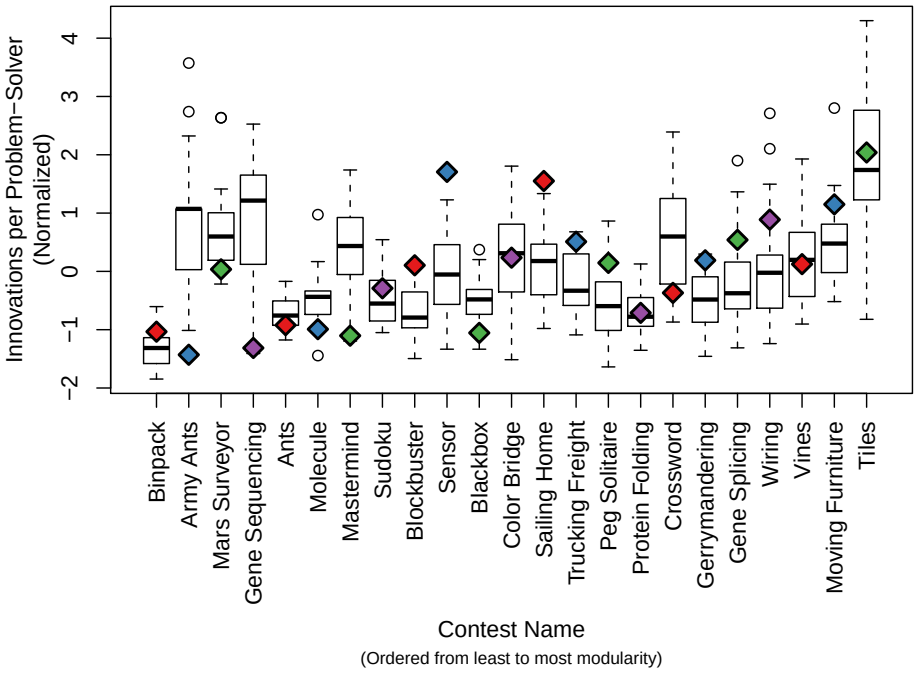
The boxes in the plots provide an indication of the range of values gathered from the 20 runs of the model. For those unfamiliar with boxplots, I will quickly review them. The thick line in the middle of the box indicates the median value. The box goes from the first and third quartiles. The whiskers extend from the box by up to 1.5 times the inner quartile range. Any points drawn outside of the whiskers are considered outliers.

The data were normalized (or also referred to as standardized) to make the empirical and model results comparable. The following equation was used for normalization:

$$normalize(X) = \frac{X - \mu}{\sigma} \tag{3.5}$$

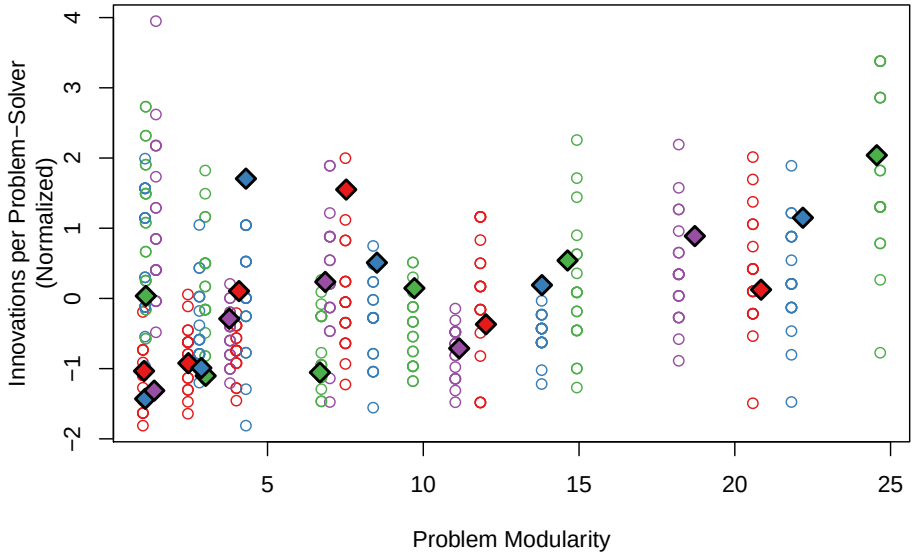


(a) Adjacent Landscape

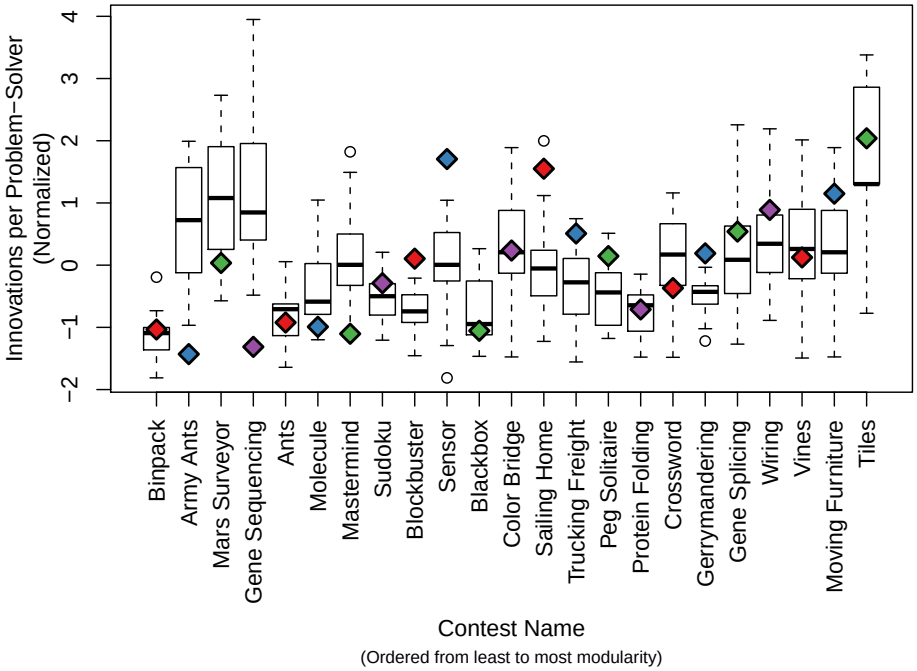


(b) Random Landscape

Figure 3.2: Results for the Monte Carlo model for recreating the programming contests. These two landscapes are the ones included in the Kauffman's original model [49].

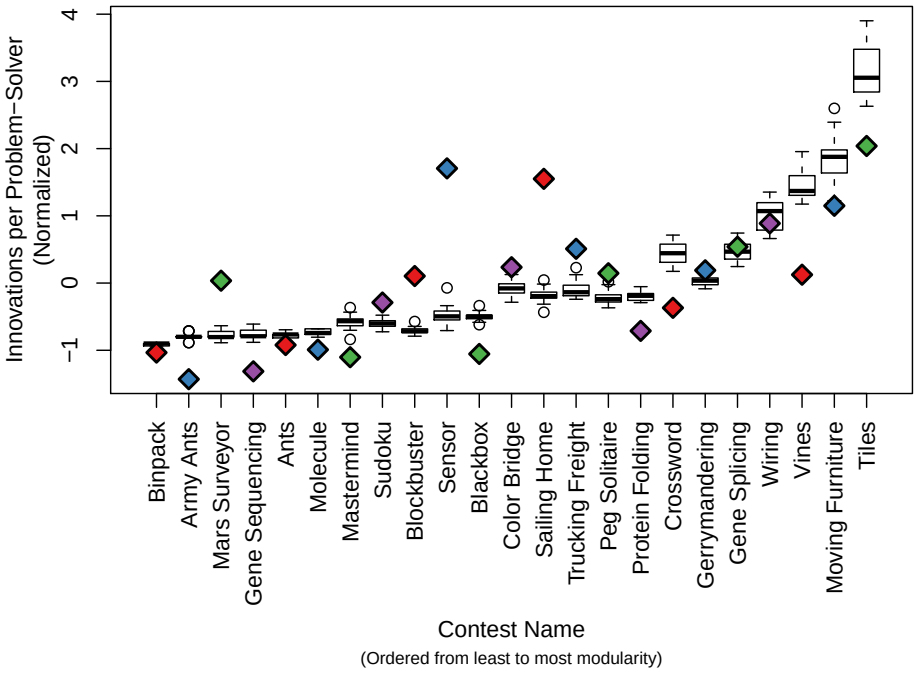


(c) Modular Landscape (Scatterplot)

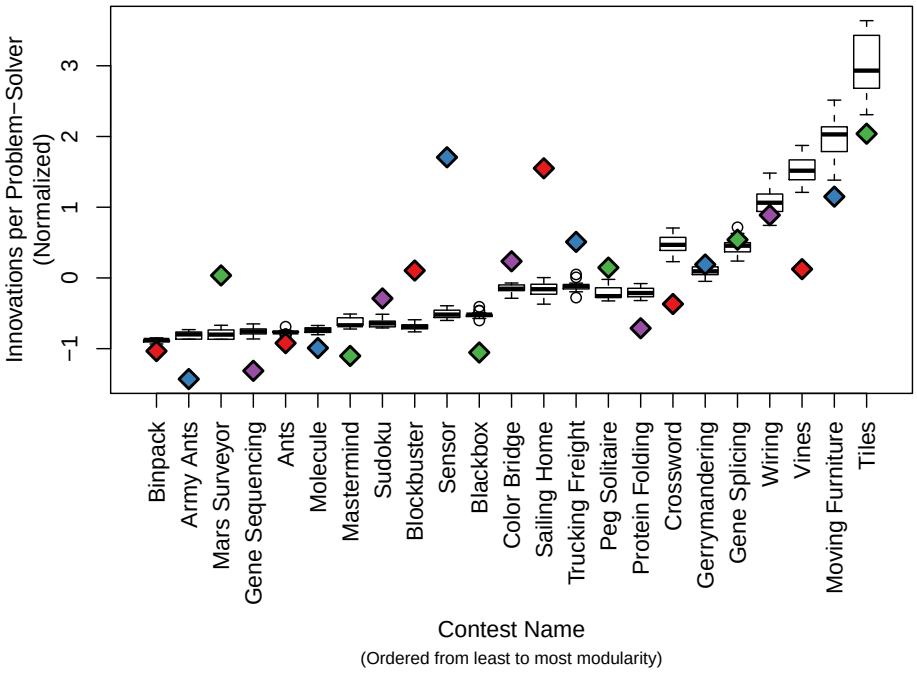


(d) Modular Landscape (Boxplot)

Figure 3.2: Results for the Monte Carlo model for recreating the programming contests. This landscape consists of blocks of genes that make up modules.

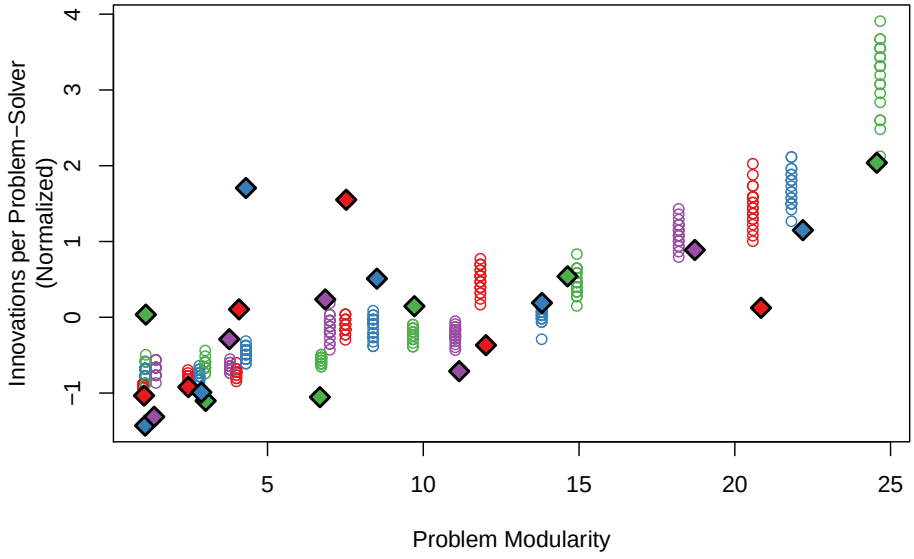


(a) Adjacent Landscape

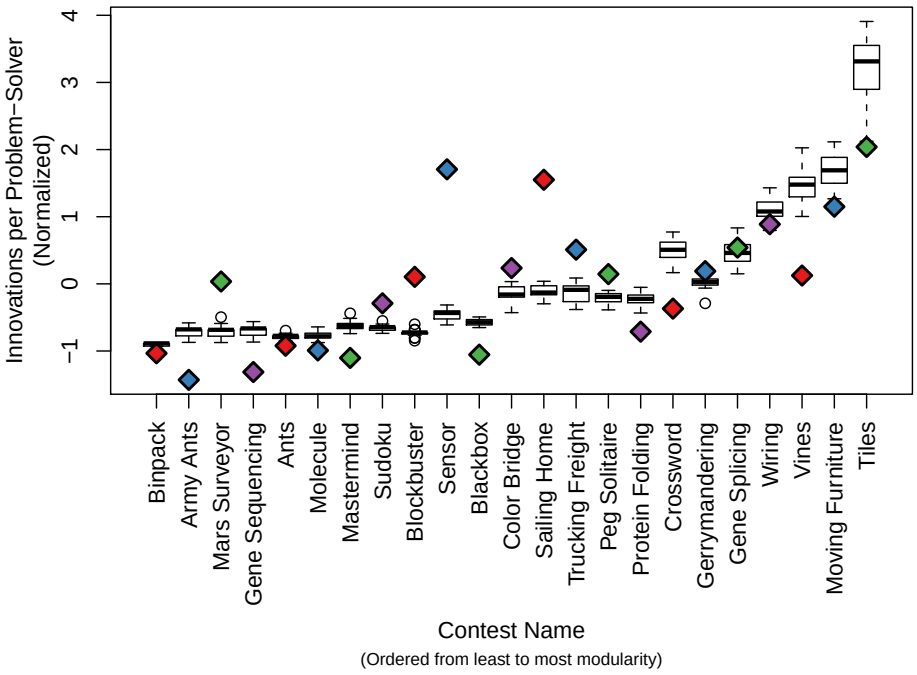


(b) Random Landscape

Figure 3.3: Results for the parallel NK model for recreating the programming contests. These two landscapes are the ones included in the Kauffman's original model [49].



(c) Modular Landscape (Scatterplot)



(d) Modular Landscape (Boxplot)

Figure 3.3: Results for the parallel NK model for recreating the programming contests. This landscape consists of blocks of genes that make up modules.

This equation resets the mean to zero and the values become the number of standard deviations from the mean.

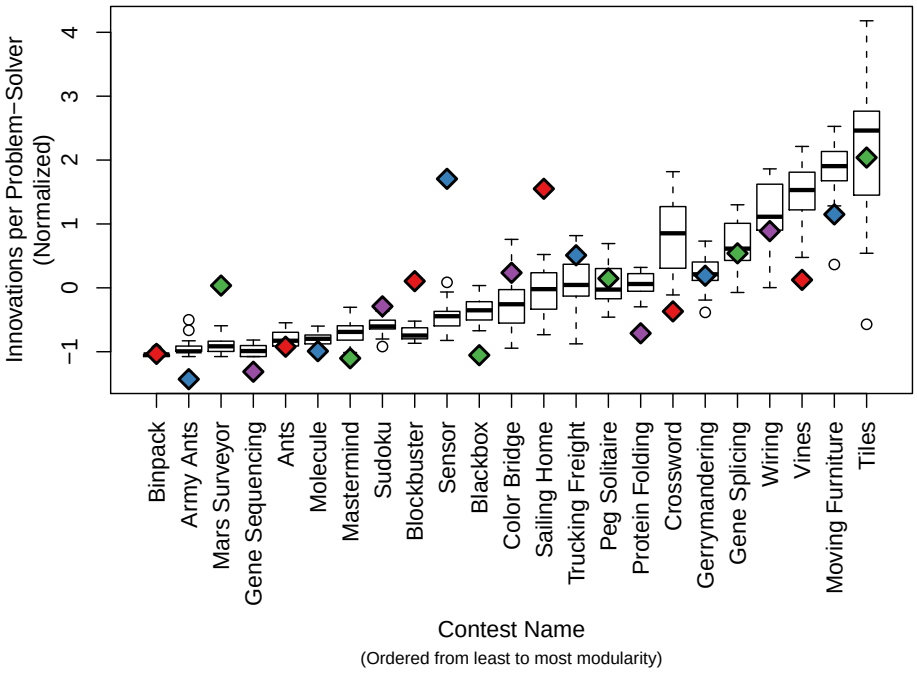
Figure 3.2 shows the results from the randomly behaving Monte Carlo model. This model appears to follow the empirical data rather well considering that at each time step each agent is creating a totally random solution without using any previously acquired knowledge about the landscape. This isn't totally surprising since the problem landscape was configured based on information from the programming contests. Furthermore, a more rigorous statistic discussed later will provide a better metric of comparison.

Figure 3.3 shows the results for the parallel NK model where multiple hill-climbers are searching the landscape. This model is the most similar to Kauffman's original formulation of the NK model. As Kauffman also found in his research [49], it is surprising how little difference there is in the results of the various landscapes. This provides an indication that for this model the most important characteristic of the landscape is the amount of emphasis and not the arrangement of the epistatic relationships. However, this is most likely not true for human problem-solvers where keeping epistatic relationships together is of importance. It has been proposed that breaking problems into subtasks is important for humans because of limitations in short-term memory [97]. This would make it particularity important for the epistatic relationships to be close to one another to limit additional burden on short-term memory.

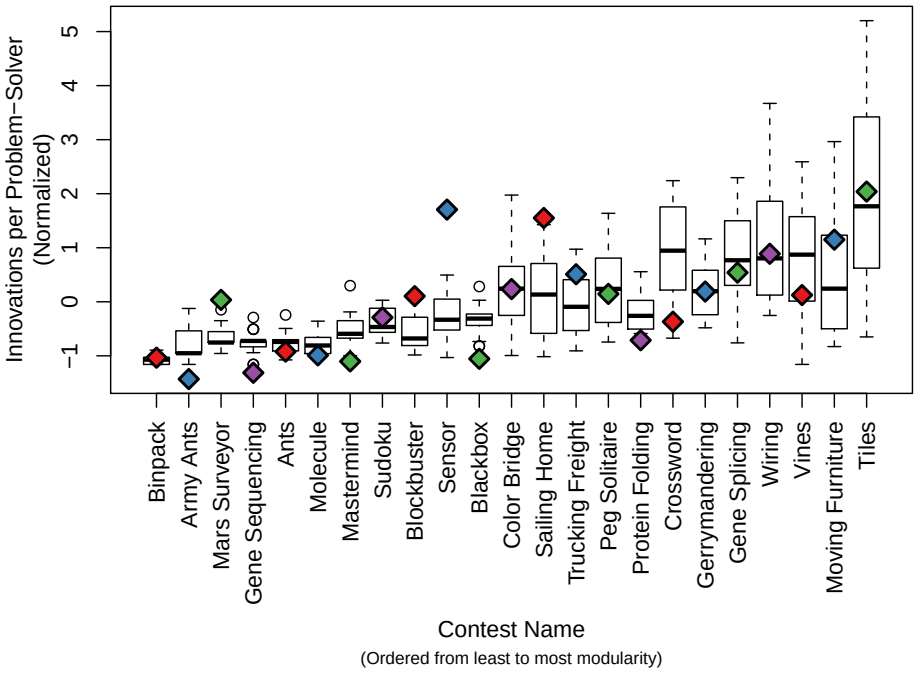
Figure 3.4 shows the results for the genetic algorithm model. Note that the agents in the genetic algorithm model were not purposefully searching for better solutions, but rather randomly recombining others' solutions. This process is not as blind as it may first appear, since it was being guided by selection.

Selection pressure seemed to have little effect on the results from the genetic algorithm model. Figure 3.5 shows the results for the modular landscape using the other two levels of selection pressure.[1] All three levels of selection pressure tend to have a similar pattern and it does not seem that the results are overly sensitive to the selection pressure used.

[1] Additional results regarding the other two levels of selection pressure are available in Appendix B.

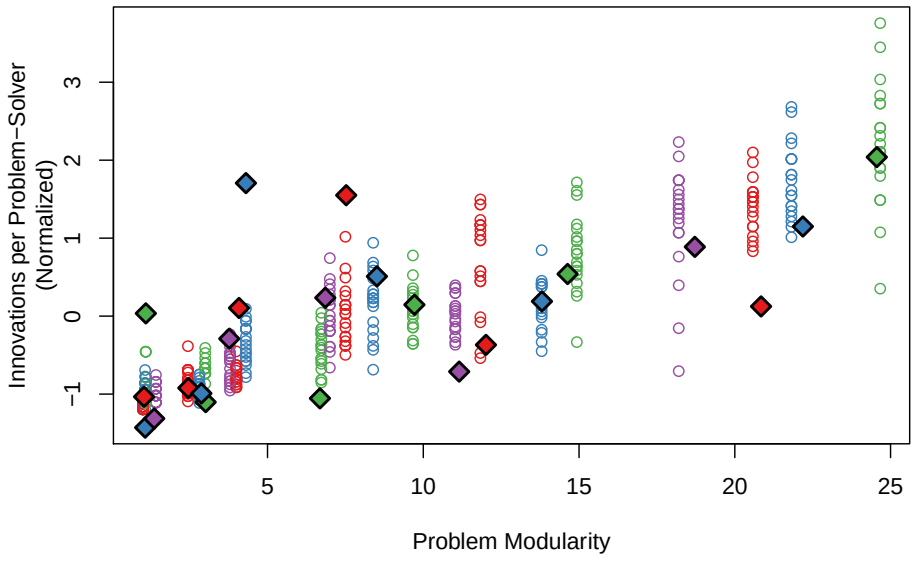


(a) Adjacent Landscape

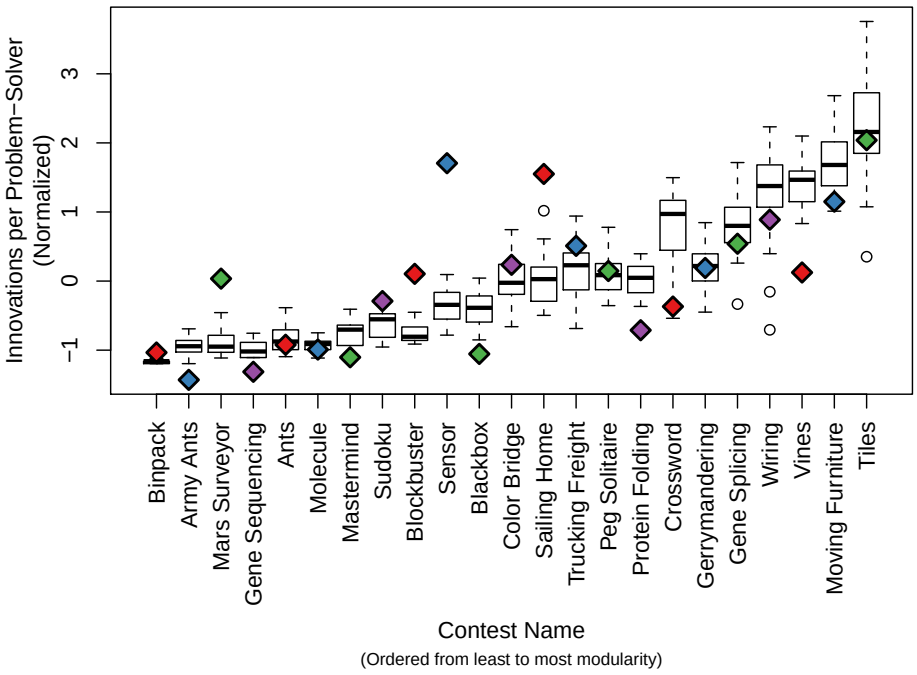


(b) Random Landscape

Figure 3.4: Results for the genetic algorithm model for recreating the programming contests. These two landscapes are the ones included in the Kauffman's original model [49].

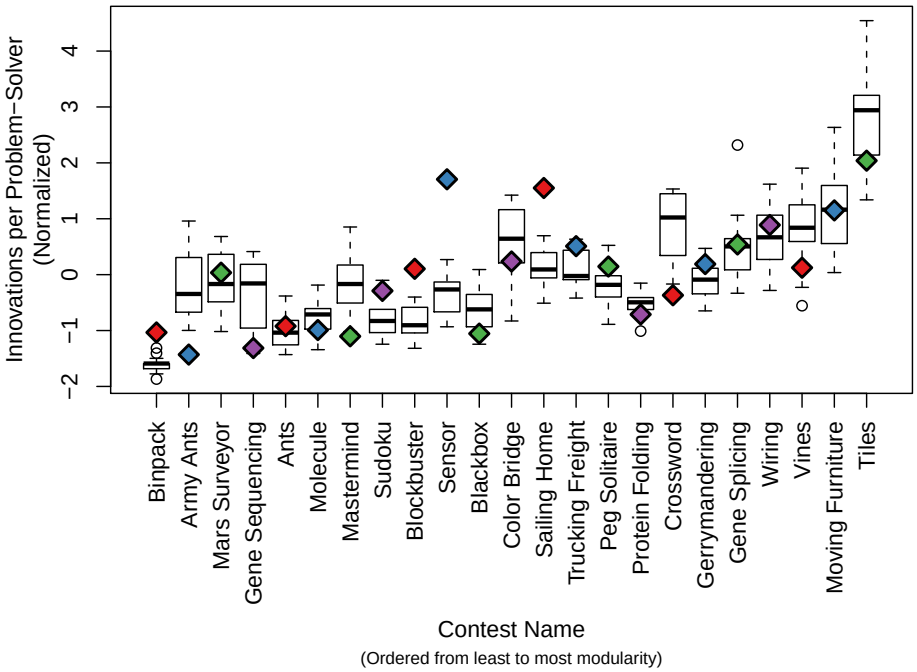


(c) Modular Landscape (Scatterplot)

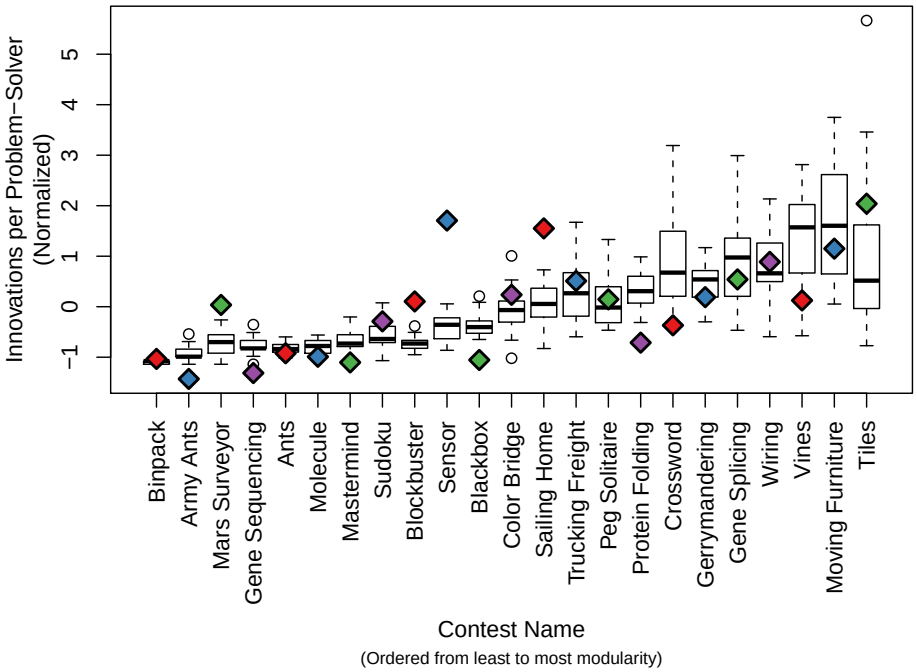


(d) Modular Landscape (Boxplot)

Figure 3.4: Results for the genetic algorithm model for recreating the programming contests. The tournament size for selection was 7, the middle value collected..



(a) Modular Landscape (Tournament Size = 2)



(b) Modular Landscape (Tournament Size = 12)

Figure 3.5: Results when modifying the selection pressure in the genetic algorithm model.

The variance of the model results vary, so any statistical test that examines if the empirical values lie within the model results will favor models that generate the most variation. Therefore, the median value for each NK landscape configuration (23 total) was used for the comparison across model results. R^2 values were calculated to determine the amount of variance in the empirical data that is explained by the model. The R^2 value can range from 0 to 1, where 1 means that the model explains all of the variance in the data. Statistical significance was calculated using linear regression to determine if the model results were a significant predictor of the empirical results.

Table 3.2 shows the results of the comparison. It is clear that the randomly behaving Monte Carlo model did not perform well. It explained little of the variance from the empirical data and is not statistically significant. However, both the parallel NK and genetic algorithm models are statistically significant and generally explains more variance than the multiple linear regression model.

Table 3.2: Comparison between computational model results and empirical data from programming contests (measured using R^2 values).

	Amount of Variance Explained (Measured with R^2 Values)		
	Landscape Type		
Computational Model	Adjacent	Random	Modular
Monte Carlo	0.032	0.050	0.111
Parallel NK	0.419^{***}	0.414^{***}	0.424^{***}
Genetic Algorithm			
Tournament Size = 2	0.502^{***}	0.218^{*}	0.399^{**}
Tournament Size = 7	0.423^{***}	0.433^{***}	0.444^{***}
Tournament Size = 12	0.327^{**}	0.475^{***}	0.319^{**}
*** $p < 0.001$ ** $p < 0.01$ * $p < 0.05$			

For comparison sake, Table 3.3 displays the multiple linear regression results using both problem modularity (M) and size (N) as independent variables and innovations per participant as the dependent variable. Just to briefly summarize this regression model, the

results show that M remains significant while N loses statistical significance. Moreover, the coefficient for N was positive when used as the only independent variable, but is now negative when M is included. The R^2 value for the multiple linear regression model is 0.398. The computational models improved upon the explanatory power of this regression model, but this is not the main advantage of the computational models. As discussed in detail later in Section 3.3, the computational models provide a platform to remove the idiosyncratic properties of the system under study and to generate a more comprehensive data set using a systemic parameter sweep.

Table 3.3: Multiple regression results using size and modularity as independent variables.

	Dependent Variable: Innovations per Participant
Independent Variables:	
Average Number Functions (M)	
Coefficient	**0.123**
p-value	**0.015**
Average Node Count (N)	
Coefficient	-1.0E-04
p-value	0.457
Intercept	1.014
R^2_{emp}	0.398
F-statistic	6.598
p-value	**0.006**

Note: Values in **bold** are significant at the 5% level.

3.2.6 Effect of Arrangement of Epistatic Relationships

The three landscapes, adjacent, random, and modular, seemed to have little effect on the computational models' ability to match the empirical data, which mimics Kauffman's observation [49]. This is quite surprising, especially in the genetic algorithm model since the crossover operator was used. Wright et al. [92] suggested that crossover would perform well

on adjacent landscapes, but not on random landscapes due to the way crossover preserves blocks of neighboring genes. Indeed, from looking at the amount of innovation per agent in Figure 3.6, the genetic algorithm agents perform significantly better on the adjacent landscape than the random landscape. As expected, the genetic algorithm agents performed best on the modular landscape, but their performance on the adjacent landscape is quite similar. Moreover, there is no statistically significant difference between the adjacent and modular landscapes.

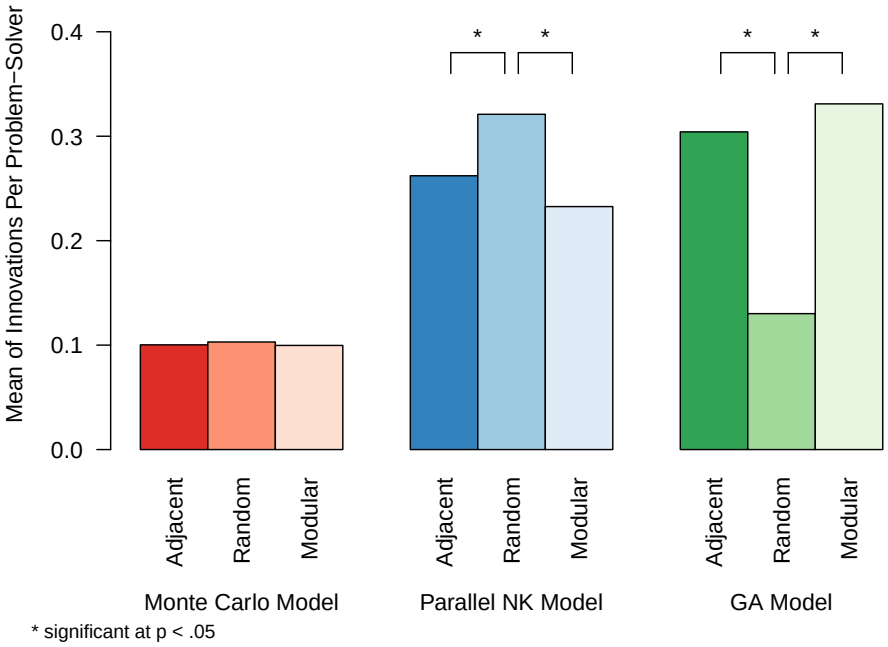


Figure 3.6: Comparison of innovations per participant for the three computational models, Monte Carlo, parallel NK and genetic algorithm models. Due to the random behavior of the Monte Carlo model, the landscape appears to make no difference in innovation performance. The genetic algorithm results are from the middle level of selection pressure (tournament size = 7). As expected, the genetic algorithm model does significantly better on the modular and adjacent landscapes, although surprisingly, the parallel NK model does best on the random landscape. Significance was determined using the Kolmogorov-Smirnov test.

To gain a better understanding of the effect of crossover on each of the landscapes, the

expected number of fitness components that could potentially change was calculated. The equations show the effect of one-point crossover, which is the type of crossover used in the model results presented here. Also, I will be referring to the number of fitness component values that could potentially change. This could be different from the number of fitness component values that actually change during the crossover operation. For example, if two genomes are exactly the same and they are used for reproduction, then the crossover operator cannot change any of the fitness component values of the newly created individual.

In the adjacent landscape, if the crossover point is not at the start or end of the genome, there are at least K fitness components that could potentially change. If the crossover point is K genes from the first or last gene, then there are $2K$ fitness components that could potentially change. The first K fitness components are due to the split of the crossover and the second K components due to the circular nature of the landscape. Therefore, the expected number of changes in fitness component values follows the equation:

$$\begin{aligned} E(\Delta C_{adjacent}) &= 2K - \left(\frac{2\,(K-1)}{N+1}\right)\left(\frac{K}{2}\right) - \left(\frac{2}{N+1}\right)2K \\ &= 2K - \frac{K\,(K+3)}{N+1} \end{aligned} \tag{3.6}$$

Where the ΔC variable is the number of fitness components that could potentially change during a one-point crossover. The equation begins with the maximum number of components that could change, $2K$, during crossover. The subtracted quantities in the equation represent the crossover locations that are near enough to the first or last gene to necessitate a reduction in the number of potential changes. For example, if the crossover location is before the first gene or after the last gene, then there will be no fitness component value changes. This eliminates double counting the fitness components for the expected number of changes.

The expected number of changes in fitness components for the random landscape is

much different:

$$E(\Delta C_{random}) =$$

$$N - \frac{\left(\sum_{L=K+1}^{N-1} L\left(1-\frac{K}{N-1}\right)^{N-L}\right) + \left(\sum_{R=1}^{N-(K+1)} (N-R)\left(1-\frac{K}{N-1}\right)^{R}\right) + 2N}{N+1} \tag{3.7}$$

This equation subtracts the expected number of fitness components that do not change from N. The L and R variables represent the possible locations of the crossover point. The crossover point could be from 0 to N where a crossover point of i means the crossover occurs immediately after the i^{th} gene. In Kauffman's landscape, the i^{th} gene always influences the i^{th} fitness component (the diagonal epistatic relationships). Therefore, if the crossover is at i, in order for the fitness component to be guaranteed to stay the same, then all of the epistatic relationships must be to the left of the crossover point ($< i$) for fitness components 1 to i and to the right of the crossover point ($> i$) for fitness components $i+1$ to N. The first summation calculates the expected number of fitness components that have all of its epistatic relationships to the left of the crossover location, and thus will not be affected by the crossover operation. The second summation does the same calculation for the epistatic relationships to the right of the crossover point. The final value in the numerator ($2N$) adjusts for when the crossover point is at the very beginning or end of the genome, thereby the crossover operation would not change any fitness components.

The effect of one-point crossover on the modular landscape is similar to the adjacent landscape, but has an upper bound of K instead of $2K$:

$$E(\Delta C_{modular}) = \frac{(N-M)\frac{N}{M}}{N+1} = K\frac{N}{N+1} \tag{3.8}$$

The M variable is the number of modules in the landscape (defined in Equation 3.2). In the modular landscape, the fitness component values do not change if the crossover point is

between two modules. However, the overall fitness of the new individual could still change because it is a new combination of the parents' fitness component values.

Crossover operates very differently on the various fitness landscapes. For the random landscape, one-point crossover can change all of the fitness component values, but at most $2K$ and K values for the adjacent and modular landscapes, respectively. Using the $E(\Delta C)$ equations from above, Table 3.4 shows the expected number of fitness components that could change during a one-point crossover operation for each of the 23 model configurations (one for each programming contest). This table makes it clear that crossover affects the fitness component values much more on the random landscape than either the adjacent or modular landscapes.

Table 3.4: Expected number of potential changes in fitness component values due to one-point crossover.

Contest Name	N	K	$E(\Delta C_{adjacent})$	$E(\Delta C_{random})$	$E(\Delta C_{modular})$
Tiles	222	8	15.6	173.6	8.0
Vines	391	18	35.0	350.1	18.0
Crossword	213	17	32.4	189.5	16.9
Sailing Home	120	15	27.8	105.2	14.9
Sensor	155	35	61.5	146.4	34.8
Color Bridge	98	13	23.9	84.2	12.9
Army Ants	46	42	43.8	43.9	41.1
Wiring	273	14	27.1	236.9	13.9
Gene Splicing	209	13	25.0	179.4	12.9
Peg Solitaire	145	14	26.4	125.9	13.9
Blackbox	148	21	38.6	134.7	20.9
Blockbuster	56	13	22.4	48.2	12.8
Sudoku	205	53	91.6	197.4	52.7
Ants	88	35	55.1	83.2	34.6
Moving Furniture	480	21	41.0	436.5	21.0
Gerrymandering	276	19	36.5	248.6	18.9
Trucking Freight	193	22	41.2	176.3	21.9
Protein Folding	397	35	66.7	375.0	34.9
Molecule	45	15	24.1	39.5	14.7
Mastermind	45	14	22.8	39.1	13.7
Gene Sequencing	17	11	13.4	14.3	10.4
Mars Surveyor	72	65	69.5	69.8	64.1
Binpack	12	11	10.2	8.3	10.2

3.2.7 Analysis Summary

Clearly, in the empirical data, innovation does not simply increase with the amount of modularity, there is a more complex relationship between problem characteristics and innovation. As seen in the previous chapter, modularity was found to be the most important predictor of innovation (using linear regression), but it is not the only predictor. Furthermore, from these model results, it is starting to become more clear that the linear assumption made in the previous chapter does not hold and there are some nonlinear relationships at work. This more complex relationship between problem characteristics and innovation will be explored later in this chapter in Section 3.4.

The Monte Carlo model did not provide much explanatory power, but it provided an important baseline for comparison with the other two computational models. It also showed that the results are falsifiable and that not any model of problem-solving would produce the same results. However, the usefulness of the Monte Carlo model is limited for the remainder of the chapter, so it will be dropped from the rest of the analyses. In the next section, the results of a parameter sweep of the parallel NK and genetic algorithm models are discussed to show how a computational model can be used to gain a clearer picture of the innovation process.

3.3 Parameter Sweep of Problem Characteristics

In the previous analysis, the computational models were validated with empirical data and both the parallel NK and genetic algorithm models provided explanatory power that was statistically significant. In this analysis, a systemic parameter sweep provides a more complete picture of the influence of problem characteristics on innovation. Instead of depending on the idiosyncratic parameter settings of the programming contests, a parameter sweep systemically increments through parameter values. Since both the parallel NK and genetic algorithm models provided a reasonable fit to the empirical data, both of these models are used in the parameter sweep. Also, the parameter sweep uses the modular NK landscape

because the modularity of the landscape is easily quantified. As noted earlier, the epistasis is an important factor for the computational models and is a well-defined quantity, but relating the results back to the real-world (i.e., the programming contests) requires the analysis to be done with regard to modules (i.e., functions in the software submissions), which can be done with the modular landscape. For the genetic algorithm model, the middle level of selection pressure (tournament size = 7) is used for the parameter sweep.

The N and K parameter settings were based on Kauffman's original values [49] with additions (especially for the larger values).

$$N \in \{8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 256, 384, 512\} \tag{3.9}$$

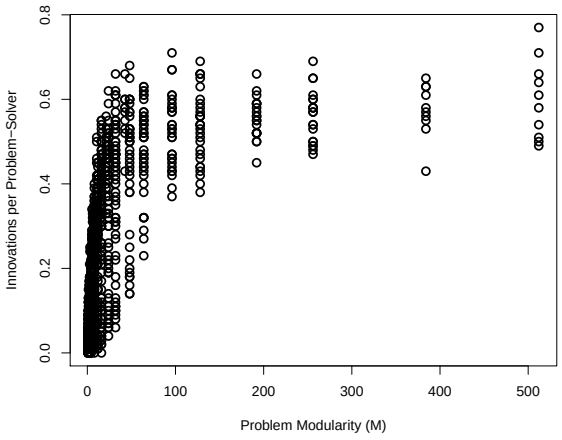
The N values include the powers of two and the values halfway between the powers of two. The values of N cover the full range of N values that were used in the previous section to validate the computational models (see Table 3.4 for the values of N and K used there). The K values are one less than the N values because the total epistasis is equal to $K + 1$.

$$K \in \{0, 1, 3, 7, 11, 15, 23, 31, 47, 63, 95, 127, 191, 255, 383, 511\} \tag{3.10}$$

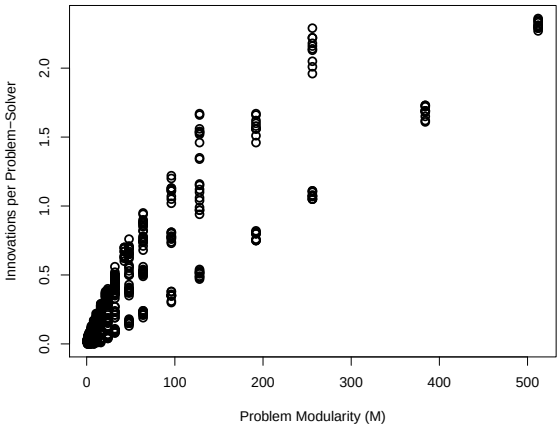
All pairs of N and K values were used in the parameter sweep where $K < N$. Each parameter setting was executed 10 times. The number of modules, M, varied from 1 to N and can be calculated using Equation 3.2.[2]

Figure 3.7 shows the relationship between problem modularity and the innovations per (artificial) problem-solver. Using linear regression, problem modularity is a significant predictor of innovation in both models ($p < .001$). However, the effect size of problem modularity is higher for the parallel NK model ($R^2 = .82$) than the genetic algorithm model ($R^2 = .34$). Also, as presented in the previous chapter, the R^2 value = .38 for the empirical

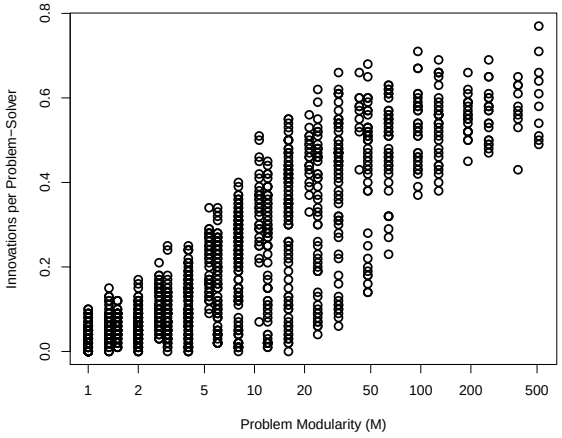
[2]It is possible to have a fraction of a module using these parameter settings of N and K. For example, if $N = 12$ and $K = 7$, then M equals $1\frac{1}{2}$. Allowing for fractional modules seemed like a more reasonable option than rounding to the nearest number of modules. The values of N and K were chosen to reduce the number of fractional modules while still obtaining a variety of M values.



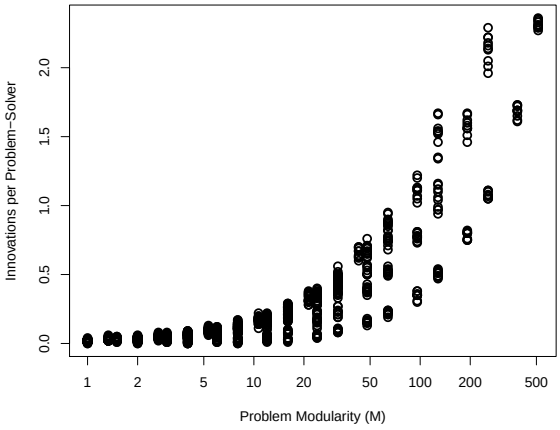
(a) Genetic Algorithm Model



(b) Parallel NK Model



(c) Genetic Algorithm Model - log(X-axis)



(d) Parallel NK Model - log(X-axis)

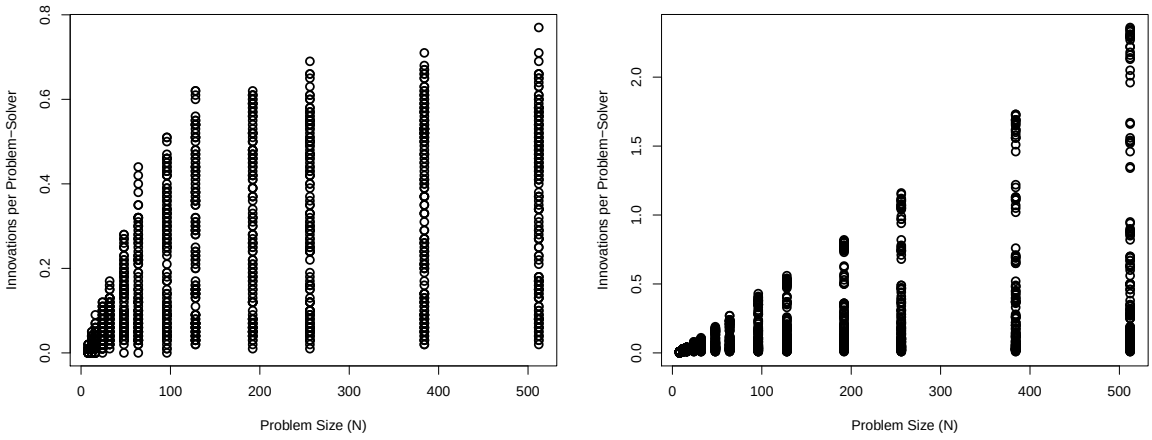
Figure 3.7: Problem modularity versus innovations per participant during the parameter sweep. This figure shows all 10 runs for each parameter setting of N and K.

data. From looking at Figure 3.7a, it is clear that there is not a simple linear relationship between modularity and innovation in the genetic algorithm model. For the genetic algorithm model, the benefit from each additional module diminishes, while the parallel NK model follows a linear relationship between modularity and innovation. This lack of linearity will be further explored later in this chapter in Section 3.4.

Figure 3.8 shows a similar story for the relationship between problem size and innovation. Again, problem size is a significant predictor of innovation for both models using linear regression ($p < .001$). The effect size for the genetic algorithm model is slightly higher than the parallel NK model ($R^2 = .27$ for the genetic algorithm model, $R^2 = .21$ for the parallel NK model, and from the previous chapter, $R^2 = .18$ for the empirical data). But again, there is an indication that the genetic algorithm model has diminishing improvements with larger problem sizes, while the parallel NK model has linear improvements.

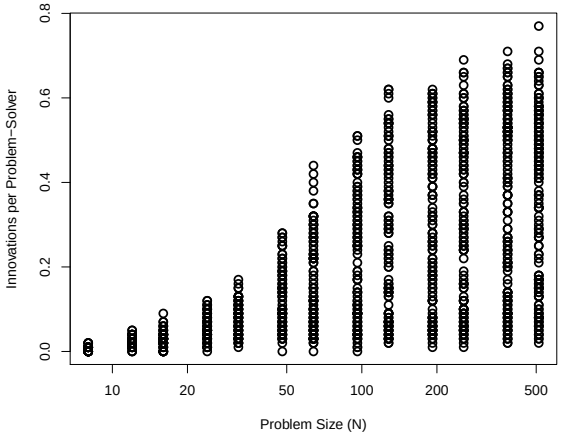
To gain a better understanding of the relationship of size and modularity, the median result for each N and K parameter setting was calculated and displayed in Figures 3.9 and 3.10. This reduced the number of points in the figures by 90% and made it possible to encode more information into the plots. Figure 3.9 shows modularity versus the median innovations per participant. Each data point is also binned according to problem size and is displayed with a different color and shape of plot marker. Similarly, Figure 3.10 shows size versus the median innovations per participant and each data point is binned according to modularity. These figures permit the display of both modularity and size in relation to innovation at the same time. Also, it more clearly shows the nonlinear relationships within the data.

Problem modularity and size both have an influence on the innovations per problem-solver, but from the genetic algorithm model it does not appear to be the simple linear relationship that was assumed in the previous chapter. The parallel NK model shows more of a linear relationship with the problem characteristics, modularity and size, and the innovations per problem-solver. Now, returning to the empirical data, similar figures can be made for the programming contest data (see Figures 3.11 and 3.12).

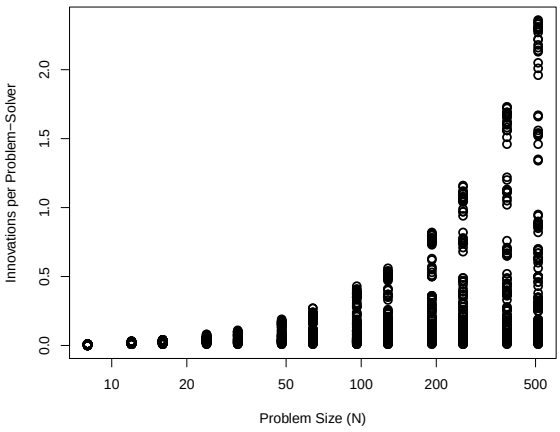


(a) Genetic Algorithm Model

(b) Parallel NK Model

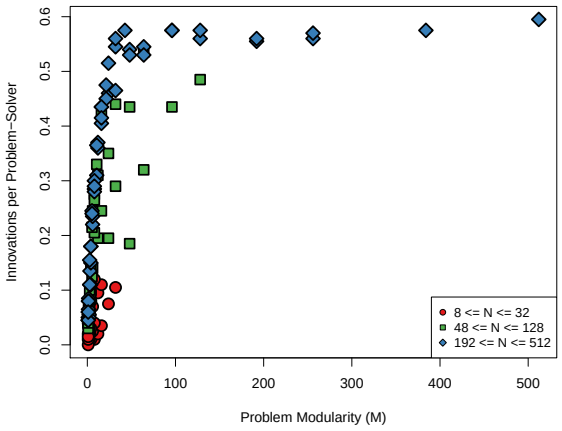


(c) Genetic Algorithm Model - log(X-axis)

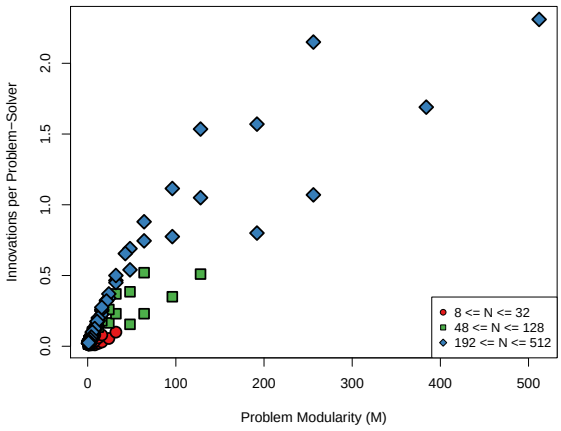


(d) Parallel NK Model - log(X-axis)

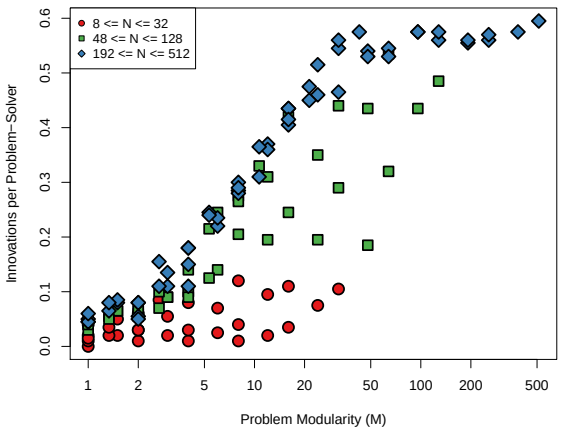
Figure 3.8: Problem size versus innovations per participant during the parameter sweep. This figure shows all 10 runs for each parameter setting of N and K.



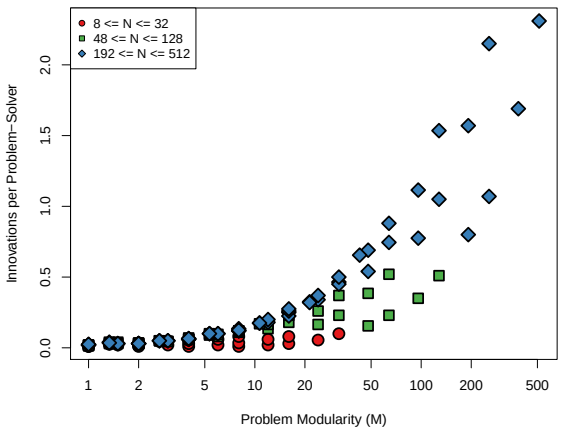
(a) Genetic Algorithm Model



(b) Parallel NK Model

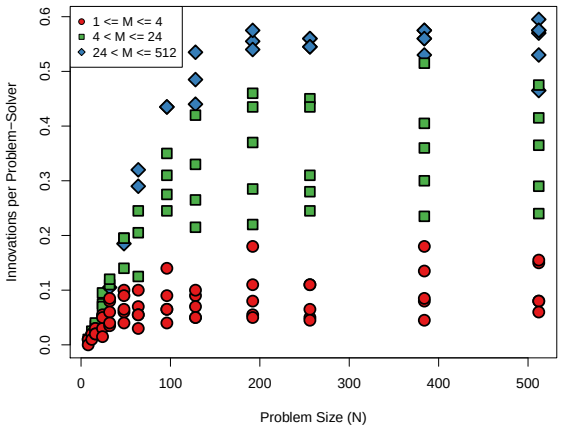


(c) Genetic Algorithm Model - log(X-axis)

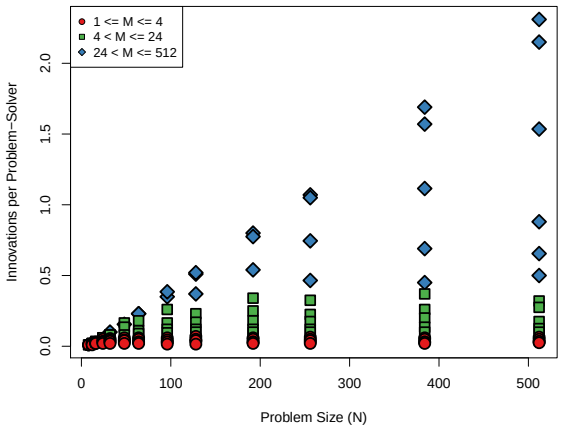


(d) Parallel NK Model - log(X-axis)

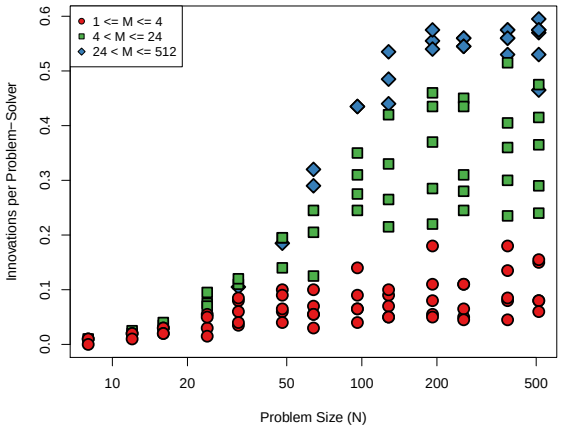
Figure 3.9: Problem modularity versus innovations per participant during parameter sweep using only the median value for each parameter configuration. The data points were grouped by problem size.



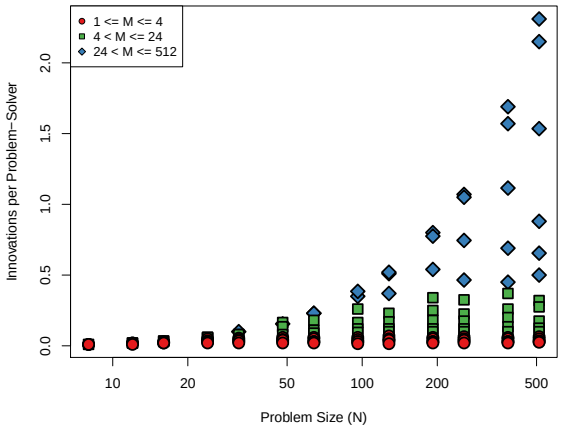
(a) Genetic Algorithm Model



(b) Parallel NK Model

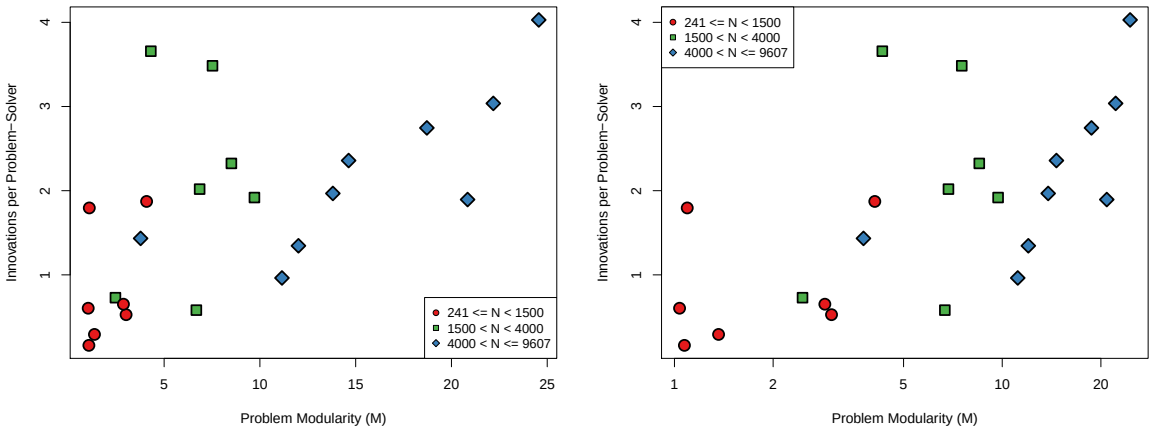


(c) Genetic Algorithm Model - log(X-axis)



(d) Parallel NK Model - log(X-axis)

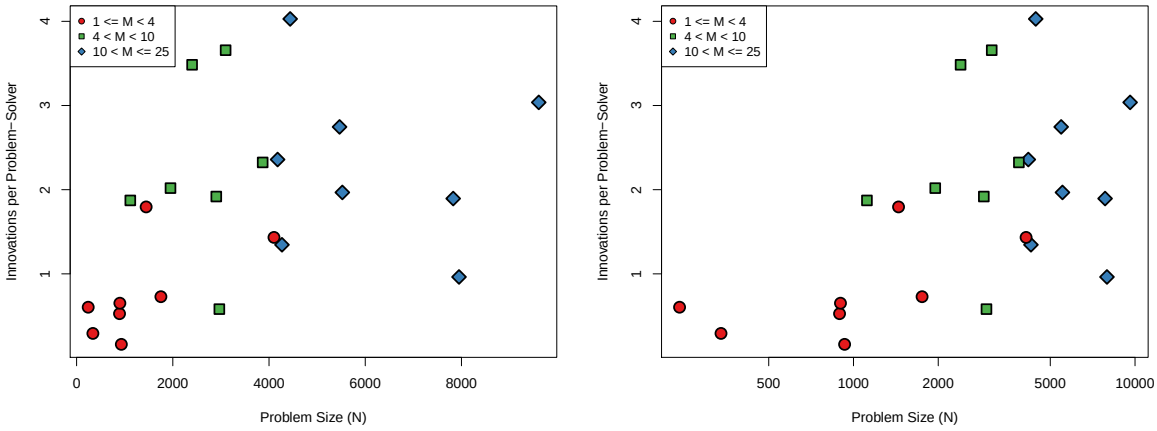
Figure 3.10: Problem size versus innovations per participant during parameter sweep using only the median value for each parameter configuration. The data points were grouped by problem modularity.



(a) Programming Contests

(b) Programming Contests - log(X-axis)

Figure 3.11: Problem modularity versus innovations per participant during the programming contests.



(a) Programming Contests

(b) Programming Contests - log(X-axis)

Figure 3.12: Problem size versus innovations per participant during the programming contests.

With only 23 data points (one for each programming contest), it is difficult to see the overall pattern of the empirical data. However, a pattern of diminishing returns from additional modularity and size seems to be a reasonable assessment. Moreover, a better model of the empirical data can be gained by using logarithmic regression instead of linear regression, where linear regression uses the equation

$$y = \alpha + \beta x \tag{3.11}$$

and logarithmic regression uses the equation.

$$y = \alpha + \beta \ln(x) \tag{3.12}$$

After moving to logarithmic regression, the R^2 values increased from .38 to .39 when using modularity as the predictor and .18 to .31 when using size. This additional prediction power comes at the expense of a more complex model, although a more complex model does not necessarily mean a less correct model. It is simply more likely to over-fit the data and suggest relationships that are not there in reality. Therefore, a move to a model with greater complexity would not generally be supported with such a small sample size (in this case, 23), but the genetic algorithm model provides additional support to move to a more complex model. Furthermore, the computational models provide better coverage of the parameter space which helps prevent over-fitting.

Starting with a linear model is a common practice to keep the model complexity low. However, the results from the genetic algorithm model suggest that a nonlinear model should be considered. In the next section, the empirical results along with the model results from the parameter sweep are used to generate a better mathematical model of innovation that is not limited by a linear assumption.

3.4 Integrating Mathematical and Computational Models

The previous results suggest that there is a nonlinear relationship between problem characteristics and innovation. Using symbolic regression the nonlinear relationships are discovered and a better mathematical model of innovation is developed. Data is analyzed using symbolic regression to develop a better mathematical model. First, mathematical formulas are generated for the empirical data from the programming contests. Second, formulas are generated for both computational models, the parallel NK model and the genetic algorithm model, using the data from the parameter sweep. Instead of presenting a single formula, several formulations on the accuracy/complexity Pareto front are included for comparison.

Figure 3.13 shows a graphical representation of the flow of the various data sets. There are two goals for the integration of the mathematical and computational models. The first goal is to use the formulas generated for the computational models to identify over-fitting of the empirical data. Due to the small sample size of the empirical data (sample size = 23), it is easy to over-fit the data and create a model that is not very generalizable. The computational models allow the generation of arbitrarily large data sets, in this case there are 130 data points. Also, with the systemic exploration of the parameter space, the parameter sweep from the computational models should not be as susceptible as the empirical data to over-fitting sparse areas of the parameter space.

The second goal is to determine which computational model provides the best formulation of the empirical data. The best formulation is judged by the predictive power and the parsimony of the formulation. As discussed in the previous sections, both the parallel NK and genetic algorithm models provided a reasonable fit with the empirical data. In this section, mathematical models of varying complexity are introduced and the predictive power is evaluated as the complexity of the mathematical models increase. These mathematical models are created by fitting the results from the computational models (the genetic algorithm model and the parallel NK model) using symbolic regression. If the computational model is a reasonable approximation of the real-world process under study, then it would be expected that the mathematical models generated by this process would continue to provide

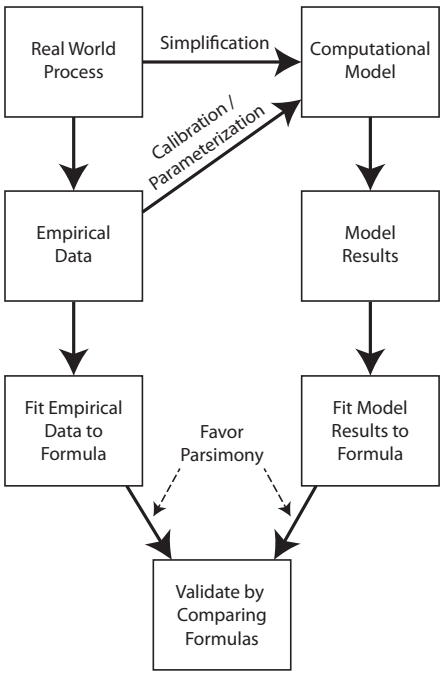


Figure 3.13: Procedure for validating computational model using fit to mathematical formula.

predictive power as their complexity grows. As the complexity of the mathematical models grow, they fit the data more closely.[3] Therefore, if the mathematical models generated by this process begin to lose their predictive power of the empirical data as their complexity grows, then this would indicate that the computational model is introducing artifacts that are not justified by the empirical data.

Both the empirical data and the model results have error in the data. The empirical data has a small sample size and any mathematical model created using this data is susceptible to over-fitting, especially mathematical models of higher complexity. In contrast, the computational model data can provide a robust data sample, but the data is created from an abstraction of the real-world process, which may introduce biases of its own. By combining the two approaches, I hope to overcome the weaknesses of the data and uncover some of the underlying complexity of the real-world process. Before moving on, I want to

[3]The mathematical model would not be on the accuracy/complexity Pareto front otherwise.

briefly introduce symbolic regression and how it generates the mathematical formulations.

3.4.1 Symbolic Regression

Symbolic regression is a technique of fitting data to a mathematical formula. I am using the Eureqa software[4] for the symbolic regression [98]. Schmidt and Lipson [98] used the software to automatically rediscover some physical laws of nature using data collected from the physical system. The software searches over various combinations of mathematical building blocks to find a formulation that best fits the data. The Eureqa software maintains a Pareto front of formulations across accuracy and complexity. The formulations' accuracy and complexity are further discussed in the next two subsections, which is followed by a brief description of the search technique used by Eureqa, genetic programming.

Formula Accuracy

The Eureqa software provides several error metrics that can be minimized during the fitting process. The results shown here were generated using two error metrics: the mean squared error (MSE) and the mean absolute error (MAE). They are expressed by the following equations:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y - f(x))^2 \tag{3.13}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y - f(x)| \tag{3.14}$$

The MSE metric assumes that noise follows a normal distribution, while the MAE metric assumes it follows a double exponential distribution. It is unknown what the underlying noise of the empirical data is, so results from both error metrics are reported. If outliers are present, then the MSE metric can be misled by placing too much importance on them causing over-fitting, while the MAE metric is more likely to under-fit the data and not

[4]Version 0.97 Beta of Eureqa Formulize was used for this analysis. The software is available for download at: http://creativemachines.cornell.edu/eureqa/

capture the nuances of the data. As seen later, the results from each of the error metrics provide a similar, although not identical story. Nevertheless, the results do not seem overly sensitive to the error metric used. This provides an indication that there are no extreme outliers that are introducing large biases.

Formula Complexity

The Eureqa software allows the user to choose the mathematical building blocks used in the symbolic regression expressions. Table 3.5 shows the mathematical building blocks used in the formulas presented here. I chose the basic mathematical building blocks along with the exponential ones. Each building block has an associated complexity and the complexity of the formula is the sum of the complexities of its building blocks. All formulas with a complexity of 10 or less are presented here. Formulas of greater complexity become difficult to interpret and are at risk of over-fitting the data. Moreover, a parsimonious model is preferred over a more complex one.

Table 3.5: Mathematical building blocks and their complexity used in the formulas.

Mathematical Building Block	Complexity
Constant	1
Integer Constant	1
Input Variable	1
Addition	1
Subtraction	1
Multiplication	1
Division	2
Negation	1
Exponential	4
Natural Logarithm	4
Power	5
Square Root	4

Genetic Programming

Genetic programming is the evolutionary computation technique used by Eureqa [98] to solve the symbolic regression problem. Genetic programming is a technique of automatically developing computer programs without explicitly telling the computer what operations to perform [99]. A program is a combination of elements from the function set and the terminal set. Elements from the function set perform some operation on input parameters, which are elements from the terminal set or function set. The terminal set can be either a constant or a domain specific input variable. The programs can be viewed as trees where branches are elements from the function set and leaves are elements from the terminal set. Genetic programming starts by randomly generating an initial population of programs. In subsequent generations through crossover and mutation, the programs evolve and through natural selection the programs become of higher fitness [99] [100]. In genetic programming, the fitness of a program is the difference between the values predicted by the program and the actual output values. The differences in actual and predict values can be aggregated several different ways and for this analysis, the MSE and MAE metrics were used.

3.4.2 Validating the Computational Models

Problem size and modularity were found to be significant predictors of innovation in the linear regression analysis in the previous chapter. Therefore, the first set of results for fitting the empirical and model data uses only size and modularity as input parameters. Table 3.6 shows the formulas generated for the empirical data gathered from the programming contests. As expected, as the formula complexity rises, the R^2 values improve. However, the formulas with higher complexity become harder to interpret and are more likely to suffer from over-fitting. For example, with a formula complexity of 10, the same input variable appears twice in both the MSE and MAE formulas, which makes the formula nonintuitive.

As discussed earlier in this chapter, Table 3.3 includes the multiple linear regression results using both M and N as independent variables. The formula complexity created by the multiple regression is 9 with a formula of $1.014 + 0.123M - .0001N$. When compared

Table 3.6: Formulas on the Pareto front for the programming contest data using size and modularity as function inputs.

Formula Complexity	MSE		MAE	
	$I = f(N, M)$	R^2_{emp}	$I = f(N, M)$	R^2_{emp}
3	$0.162M$	.38	$0.161M$	.38
5	$0.944 + 0.0978M$	.38	$\ln(M)$	.39
7	$\sqrt{0.436M}$	.40	$0.635\sqrt{M}$	.40
8	$\frac{3.46M}{5.23+M}$	.39	$\frac{987M}{1.58e3+N}$	.43
9	$\sqrt{M} - 0.00024N$	.41	*	*
10	$\frac{N+774M}{1.61e3+N}$	.48	$0.149M + \frac{0.483}{4.47-M}$	.57

* There was no formula on the Pareto front for this complexity.

to the formulas created in Table 3.6, the symbolic regression results provide only a small improvement in explanatory power (measured by R^2 values) over the multiple regression results. As noted earlier, the linear assumption does not seem like a good one, but searching for nonlinear models using only N and M provide limited improvements. This issue will be addressed further in Section 3.4.3, but now the symbolic regression results are used to further validate the computational models.

Earlier in this chapter, both the genetic algorithm and parallel NK models followed the empirical data reasonably well. The symbolic regression results can help further validate the two computational models. Just as with the empirical data, formulas were generated for the model data gathered during the parameter sweep discussed earlier. Since the parameter sweep had 10 values for every set of parameter settings, only the median value for a particular parameter setting was used during the symbolic regression procedure. If the fitting was performed on all 10 values, then it would be impossible to fit all 10 values at once because they have different output values, but the same input parameters.

Table 3.7: Formulas on the Pareto front for the genetic algorithm parameter sweep data using size and modularity as function inputs.

Formula Complexity	MSE			MAE		
	$I = f(N, M)$	R^2_{mod}	R^2_{emp}	$I = f(N, M)$	R^2_{mod}	R^2_{emp}
3*	$0.001N$	.28	.18	$0.001N$	.28	.18
5	$0.17 + 0.002M$	.36	.38	$0.09 + 0.004M$	.36	.38
6	$\frac{M}{49.7+M}$	.72	.38	$\frac{M}{36.8+M}$	.75	.38
7	$0.11 \ln(M)$	.76	.39	$0.12 \ln(M)$	.76	.39
8	$\frac{M}{24+1.7M}$	.78	.39	$0.6 + \frac{-5.6}{8.5+M}$	.77	.39
10	$\frac{M-0.27}{22.1+1.7M}$	.78	.39	$0.58 + \frac{-1264}{2262+NM}$	.93	.30

* A formula using M instead of N would give higher R^2 values.

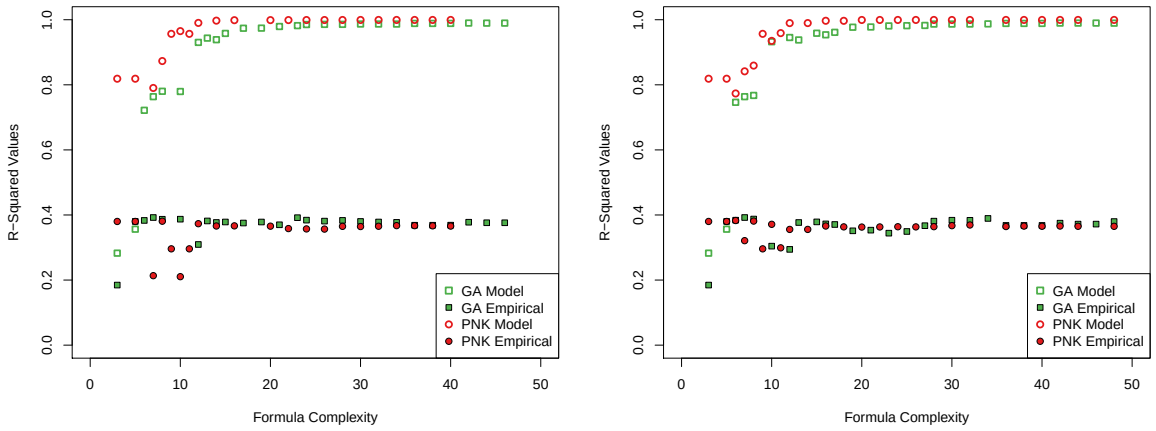
Tables 3.7 and 3.8 show the formulas generated for the genetic algorithm model and parallel NK model, respectively. In these tables, there are two sets of R^2 values. One set (R^2_{mod}) tells how much variance is explained by the formula for the model data and the other set (R^2_{emp}) tells how much variance is explained for the empirical data from the programming contests. The R^2_{emp} are the most importance values in the tables because they relate the formulas back to the real-world. From the tables, both of the computational models maintain similar R^2_{emp} values across several different formula complexities. Furthermore, Figure 3.14 shows the R^2_{emp} and R^2_{mod} values for every formula generated with a complexity of 50 or less. As the formula complexity increases, the formulas continue to have explanatory power. This is surprising because as the complexity rises, the formulas fit the parameter sweep data ever more closely and it would be expected that over-fitting would occur causing a lack of generality for the empirical data. However, this does not happen and the more complex formulas maintain their ability to provide explanatory power.

Table 3.8: Formulas on the Pareto front for the parallel NK parameter sweep data using size and modularity as function inputs.

Formula Complexity	MSE			MAE		
	$I = f(N, M)$	R^2_{mod}	R^2_{emp}	$I = f(N, M)$	R^2_{mod}	R^2_{emp}
3	$0.006M$	.82	.38	$0.008M$	.82	.38
5	$0.08 + 0.006M$	.82	.38	$0.03 + 0.008M$	.82	.38
6	*	*	*	$\frac{M}{55.1+M}$	.77	.38
7	$0.11 + 0.00002NM$	.79	.21	$0.008M + 0.0003N$	.84	.32
8	$\frac{M}{83.9+0.34M}$	.87	.38	$\frac{2.37M}{155+M}$	.86	.38
9	$0.005\sqrt{NM}$	.96	.30	$0.004\sqrt{NM}$	.96	.30
10	$\frac{NM}{18374+169M}$	.97	.21	$\frac{0.013NM}{N+M}$	.94	.37

* There was no formula on the Pareto front for this complexity.

The explanatory power of the formulas generated for each of the computational models are almost identical. Therefore, this procedure did not provide a conclusive answer to which computational model is best, but it does provide evidence that both computational models are reasonable abstractions of the real-world problem-solving process. This is particularly surprising considering that the parameter sweep data appears quite different for each of the computational models (see Figures 3.9 and 3.10). A better model of group problem-solving would most likely include elements from each of the computational models. This better model would include both the collaborative aspects of the genetic algorithm model and the purposeful exploration of the fitness landscape seen in the parallel NK model. This research provides evidence that both group collaboration and individual inquisition are both valuable in understanding and predicting the behavior of a problem-solving process.



(a) MSE

(b) MAE

Figure 3.14: The mathematical formulas continue to have explanatory power as the formula complexity increases. For both the MSE and MAE metric, the formulas for both computational models continue to have explanatory power. This is surprising considering how closely the formulas fit the parameter sweep data. It would be expected that the formulas would start to over-fit the parameter sweep data and no longer generalize to the empirical data.

3.4.3 Developing a Better Model

The improvement in explanatory power was modest when moving from a linear mathematical model to a nonlinear one. Therefore, this section explores other problem characteristics other than size (N) and modularity (M) to see if they are helpful in building a better model. Problem complexity seemed like a promising candidate, but in the linear regression analysis from the previous chapter, it was not a statistically significant predictor of innovation. Perhaps problem complexity is an important factor, but in a nonlinear way. This section reintroduces problem complexity to see if a role for it can be discovered.

In the empirical data, the problem complexity was measured for each function in every submission. Therefore, for each submission there are M complexity values where M is the number of functions in the submission. For this analysis, the problem complexity (C) is actually three different variables. The C_{max}, C_{min}, and C_{avg} variables measure the average

across submissions for the maximum, minimum, and average complexity.

Table 3.9 shows the results for the symbolic regression using problem size, modularity and complexity as input parameters. Starting at a formula complexity of 6, the term $\frac{N}{C_{max}}$ appears and causes quite a large jump in the R^2_{emp} value. This jump occurs even when the modularity variable (M) is not present in the formula, which is quite surprising given that when used individually modularity was the best predictor of innovation.

Table 3.9: Formulas on the Pareto front for the programming contest data using size, modularity and complexity as function inputs.

Formula Complexity	MSE		MAE	
	$I = f(N, M, C)$	R^2_{emp}	$I = f(N, M, C)$	R^2_{emp}
3*	$0.162M$	.38	$0.161M$	.38
4	$\frac{4.26}{C_{min}}$	.36	**	**
5*	$0.944 + 0.0978M$	.38	$\ln(M)$	.39
6	$\frac{0.0103N}{C_{max}}$	.58	$\frac{0.0103N}{C_{max}}$	.58
8	$0.59 + \frac{0.00783N}{C_{max}}$	.58	$\frac{0.0143N-M}{C_{max}}$	.59
10	$\frac{N}{47C_{max}+0.182N}$	.61	$\frac{0.0148N-1.08M}{C_{max}}$	.59

* These formulas are the same as those found earlier in Table 3.6.
** There was no formula on the Pareto front for this complexity.

The formulation of $\frac{N}{C_{max}}$ seems reasonable, but this result is based on only 23 data points. The parameter sweep data from the computational models are used to reinforce this finding. The computational models did not include problem complexity explicitly, but K provides a similar variable. As discussed in the previous chapter, cyclomatic complexity is the the number of independent linear pathways [86] in a piece of software and was used to

measure the problem complexity of the MATLAB programming contest submissions. The K variable defines the number of epistatic relationships there are in the fitness landscape. As the number of the epistatic relationship increases, the problem complexity would also increase. While K may not be a perfect analogy to problem complexity, it seems like a close proxy. Furthermore, while using the NK model to study social networks within the open source software community, Wagstrom and collaborators [101] found a similar result where in highly coupled projects (i.e., high K value), the rate of project improvements decreases.

Tables 3.10 and 3.11 show the symbolic regression results for the computational models using the variables N, M, and K. The genetic algorithm and parallel NK data both generated formulas that include a term where N is divided by K. Since the C_{max} and K variables are measuring two different things, the R^2_{emp} values could not be calculated. But, the results from the computational models provide additional support that $\frac{N}{C_{max}}$ is a reasonable predictor of innovation. This makes for an intuitive model because larger simple problems should have many improvements while smaller complex problems should have few improvements.

Table 3.10: Formulas on the Pareto front for the genetic algorithm parameter sweep data using N, M, and K as function inputs.

Formula Complexity	MSE		MAE	
	$I = f(N, M, K)$	R^2_{mod}	$I = f(N, M, K)$	R^2_{mod}
3*	$0.001N$	.28	$0.001N$	.28
5*	$0.17 + 0.002M$	.36	$0.09 + 0.004M$	.36
6*	$\frac{M}{49.7+M}$	.72	$\frac{M}{36.8+M}$	.75
7*	$0.11 \ln(M)$	.76	$0.12 \ln(M)$	.76
8*	$\frac{M}{24+1.7M}$	.78	$0.6 + \frac{-5.6}{8.5+M}$	.77
10	$\frac{N}{162+N+17K}$	.94	$\frac{N}{138+N+18.3K}$	.94

* These formulas are the same as those found earlier in Table 3.7.

Table 3.11: Formulas on the Pareto front for the parallel NK parameter sweep data using N, M, and K as function inputs.

Formula Complexity	MSE		MAE	
	$I = f(N, M, K)$	R^2_{mod}	$I = f(N, M, K)$	R^2_{mod}
3*	$0.006M$	.82	$0.008M$	.82
5*	$0.08 + 0.006M$	.82	$0.03 + 0.008M$	.82
6*	**	**	$\frac{M}{55.1+M}$	.77
7*	$0.11 + 0.00002NM$	.79	$0.008M + 0.0003N$	.84
8	$\frac{0.021N}{4.5+K}$	.99	$\frac{N}{211+53K}$	.99
10	$\frac{0.021N-0.22}{4.4+K}$	.99	$\frac{N-7.6}{188+51K}$	.99

* These formulas are the same as those found earlier in Table 3.8.
** There was no formula on the Pareto front for this complexity.

3.5 Discussion

The use of models to study problem-solving in individuals is quite natural. Individuals create a model of a problem when solving it [102]. This simplifies the problem-solving process and removes many of real-world complexities of the problem [102]. Furthermore, Basalla notes that trial and error methods are the predominant element for technological change [25]. Also, Simon stated that:

> "human problem solving, from the most blundering to the most insightful, involves nothing more than varying mixtures of trial and error and selectivity." [29, pg.195]

Therefore, techniques from evolutionary computation seem like an ideal start for modeling problem-solvers.

Popper also notes the importance of trial and error in the problem solving process, but also states that learning is an important part of the process [103]. No doubt learning is important to problem-solving, but in this chapter, simple computational models that did not include any learning were able to provide reasonable levels of explanatory power. Furthermore, the computational models (i.e., the genetic algorithm model and the parallel NK model) were used to develop a better model of innovation. This also included validation of the computational models in two ways. The first validation compared the normalized results from the computational models to the normalized empirical data. This validation provided support that the genetic algorithm model, which allowed for collaboration between problem-solvers, and the parallel NK model, which had problem-solvers work independently, both provided statistically significant predictions of the empirical data. The Monte Carlo model showed that not all computational models were capable of generating statistically significant predictions and showed that the experiment was falsifiable. Also, by using various arrangements of epistatic relationships (random, adjacent, and modular landscapes), Kauffman's observation [49] was reiterated where the arrangement of epistatic relationships did not matter as much as expected. However, this would most likely not be true for humans

where the arrangement of epistatic relationships would be of importance.

A parameter sweep allowed for a more complete view of the state space and showed that there was a nonlinear relationship between the problem characteristics (size and modularity) and innovation. This suggested a move away from the linear mathematical model used in the previous chapter to a nonlinear one. Using symbolic regression, the empirical and computational model data was fitted to formulas of varying complexity along a Pareto front. This made it clear that a mathematical model using only problem size and modularity could provide only slightly more explanatory power than the multiple linear regression results. This led to the second technique used to validate the computational models. Using symbolic regression, formulas were generated using the parameter sweep data for the genetic algorithm model and parallel NK model. Both computational models generated formulas that maintained explanatory power throughout a wide range of formula complexities providing evidence that both computational models are capturing some aspects of the real-world process.

Finally, a better mathematical model was generated by adding problem complexity to the symbolic regression. Problem complexity was not found to be a statistically significant predictor of innovation in the linear regression analysis in the previous chapter, but was found to be an important component in the symbolic regression analysis. The discovered formula includes problem size divided by problem complexity ($\frac{N}{C_{max}}$). With a small sample size (23), it is difficult to support the introduction of a more complex model, but the symbolic regression results for the computational models provided more support for the additional model complexity.

This chapter provides an example of how to use computational and mathematically models in combination to develop a better model of the process under study. Two separate methods of validation of the computational models were also put forward. The resulting mathematical model of innovation is quite intuitive. The model suggests that large simple problems will have more innovations than smaller complex problems.

The model variations discussed in this chapter all deal with static landscapes. These landscapes are appropriate for innovation that occurs over a relatively short time period. By allowing the landscape to remain static, it allows for a much simpler model. The next chapter explores an innovation process that occurs over a long time period (100 years) where the fitness of a solution is not static and depends on the current state of the environment.

Chapter 4: Analyses of Dynamic Fitness Landscapes

In this chapter, I move away from the static fitness landscapes discussed in the previous two chapters and onto dynamic fitness landscapes. In these dynamic landscapes, an entity's fitness is not fixed and exogenous, but depends on the other entities in the environment, which could be changing over time. Therefore, the landscape is co-evolutionary. In the case of book publishing, as topics become popular, opportunities are opened up for others to write about similar subjects, new terminology arises and is introduced into the written corpus, altering the usage and usefulness of language that was used previously.

This chapter discusses quantitative analyses of word frequency data from the Google Books project. The data used here is from over 600,000 books that were published from 1900 to 2000. This data represents a truly open-ended innovation process where the usefulness of terms are in constant flux as new subject matter is introduced. This permits the study of an innovation process with a dynamic fitness landscape, the second type of innovation (i.e., innovation over multiple S-curves).

4.1 Google Books Data

This chapter discusses data analyses performed on word frequency data from the Google Books project [18]. The data used here includes the word frequency count from over 600,000 books that were published from 1900 to 2000. For each year of this 101-year period, word frequency counts were used from approximately 6,174 books per year. Therefore, over 53 billion word occurrences (5.3×10^{10}) were used in this analysis. In this data, words are the building blocks under study.

Some terms used in this chapter have been used in various contexts, so a brief introduction to these terms is due. A n-tuple is a sequence of characters of length n, where a n-gram

is a sequence of words of length n. Therefore, a 1-gram is a single word. In the Google Books data, a 1-gram could be composed of various characters, including: letters, numbers, and punctuation [70]. Google posted frequency data of n-grams going up to 5-grams [18], but this dissertation only uses the 1-gram data.

Google scanned the books using optical character recognition and data is provided for books published from 1550 to 2008. Since text from older books tend to be more difficult to recognize due to book wear or other factors, only books published since 1900 was used in this analysis. Google estimates that 98% of the words were correctly digitized for modern English books (from supplemental information of [70]).

4.1.1 Types of Analysis Performed

The first analysis explores the distribution of word usage and shows how Zipf's Law does not hold for large datasets [104] [105]. Additional analysis is performed to determine why Zipf's Law fails to explain the word frequency data. This leads to insight into how the building blocks of language (i.e., words) are structured.

The second analysis explores how word usage changed over time. Using the same approach as Stanley et al. [106], the change in word usage followed a Laplace distribution (i.e., double exponential). Stanley et al. [106] found this same pattern using firm data. The growth rates of the usage of words, the building blocks, are analyzed for the 100 year period.

4.1.2 Preprocessing of the Google Books Data

Before describing the analysis performed on the data, the preprocessing performed on the data should be noted. First, I removed all 1-grams that contained a character other than the 26 English letters (A-Z and a-z). Therefore, any 1-grams that contained numbers or punctuation[1] were removed. Since, the 1-grams that I analyzed contain only English letters, I will refer to them as words.

[1]For the most part, punctuation was included in a 1-gram by itself. For example, the sentence "He ran." contains three 1-grams.

From this data, I made all of the words lower case and combined duplicate words[2], so the data is no longer case sensitive. Then, I created two sets of data: one data set contained the original inflected forms of the words and the other contained only the words' stems. Morphology was computed using the Stanford NLP Tools (http://nlp.stanford.edu/), which based their implementation on the work of John Carroll and collaborators [107] [108]. There are two types of morphology: inflectional and derivational. Only inflectional morphology was used to discover the stem of the word. Inflection changes the word's tense (e.g. 'run' can be inflected to 'ran' or 'running') or number (e.g. 'computer' can be inflected to 'computers'). Inflection does not change the underlying semantic content of the word. Derivational morphology combines existing words to create a new word, which has a new meaning (e.g. 'uncompress' and 'neighborhood'). For this analysis, inflectional morphology was most suitable because as stated by Spencer, "inflected forms are just variants of one and the same word" [pg. 9] [109]. Table 4.1 provides examples of how inflections were stemmed.

Table 4.1: Words and their stems.

Inflections	Stem
am, are, is, was, were, be, being, been	be
have, has, had	have
haves*	haves
computer, computers	computer
compute, computed, computing, computes	compute
run, ran, running, runs	run
compress, compressed, compressing, compresses	compress
uncompress, uncompressed, uncompressing, uncompresses	uncompress
neighbor, neighboring, neighbors	neighbor
neighborhood, neighborhoods	neighborhood

* As in the 'haves' and the 'have-nots'.

Traditionally, inflected forms were used to study word frequency [110] [111] [112] and

[2]This means the words 'the', 'The', and 'THE' would be combined into one word with a single frequency.

stemmed forms for applications in stylometry, where writing style is analyzed [112], For example, Yule used frequencies of uninflected nouns to identify the author of the text [113].[3] The analyses discussed here were performed on both the inflected and stemmed forms of the words.

4.2 Analysis of Word Usage

One of the earliest[4] and most significant characterizations of building block usage was made by Zipf in describing the usage frequency of words [110] [111]. Zipf's law is a special case of power law distributions. Power laws have been found in many domains [118] [88] [116], including: income distribution [119], population of cities [111] [120] [121], firm sizes [122], number of papers authored [123], citations of papers [124], hits of websites [125] [126], link distribution of the internet [127], pages per website [128], authors per Wikipedia article [129], and intensity of wars [130] [131].

Power laws became so common a popular paper by Clauset and collaborators reexamined 24 data sets to determine if alternative distributions (e.g., log-normal and exponential) provided a better fit. Of the 24 data sets analyzed, they found that 17 were consistent with a power law. But, the word frequency data was the only data set that a power law was "truly convincing" because the data was an "excellent fit" and the alternatives did not carry any weight [88]. If there is a an underlying mechanism that unifies processes that generate power law distributions, then word frequency may be the prototypical example to study. However, as discussed later, the distribution of word frequency is not as straightforward as it first appears when examining large data sets.

[3]The search for this type of summary statistics that provides a "fingerprint" of the text have not been found to be reliable in identifying authorship [114] [115]. Modern stylometry uses artificial intelligence techniques that focus on function words (i.e. words that provide little meaning, but add structure) [115].

[4]Newman [116] notes that Estoup [117] was the first to make the observation of word frequency, but Zipf studied it at length.

Zipf's law of word frequency was originally stated as:

$$P(r) \propto r^{-1} \tag{4.1}$$

But, commonly is expressed by the following equation[5]:

$$P(r) \propto r^{-\alpha} \tag{4.2}$$

Here r is the rank of a word and the probability of a rank is proportional to $r^{-\alpha}$. In Zipf's original formulation, he stated his law with $\alpha = 1$ [111] and typically in empirical word frequency data α is approximately 1. When the distribution is plotted on a log-log scale, the distribution is a straight line with slope = -α.

There have been several theories trying to explain Zipf's Law. Zipf [111] put forward the 'Principle of Least Effort' where, in this "economy of speech" [pg. 20], the speaker and listener minimize the amount of effect required to communicate. To minimize effort, the speaker wants to minimize the lexicon size ('Force of Unification'), while the listener wants to expand it by having a unique word for each possible meaning ('Force of Diversification') [111]. The 'Force of Unification' reduces the lexicon size, but increases the average usage of each word, while the 'Force of Diversification' grows the lexicon size, but decreases the average usage of each word. These two forces create the "vocabulary balance" that is observed in Zipf's Law [111].

Mandelbrot [112] generalized Zipf's Law to allow for a better fit of empirical data:

$$P(r) \propto (r + m)^{-\alpha} \tag{4.3}$$

This is known as the Zipf-Mandelbrot Law and introduced two additional parameters (α and m). The α parameter is the same as mentioned before and the m parameter allows the

[5]Zipf's Law is a power law relationship. Power laws were also used earlier in Chapter 2, but the interpretation of those power laws are different from the ones discussed here. An overview of power laws and the reasoning for the two interpretations are included in Appendix E.

distribution to have fewer observations for the most frequent words than what is predicted by the standard Zipf's Law. Mandelbrot [112] based his work on Shannon's work on information theory. Information theory attempted to find the most compressed way of encoding a message. Mandelbrot saw language as the inverse problem, where the set of words were given and the task was to find the most efficient way to assign meaning to those words [112].

From empirical word frequency data, Zipf's Law has been found to be an acceptable fit for ranks between approximately 100 to 2000 and Mandelbrot's generalization of Zipf's Law only improves upon the most frequent words (ranks < 100) [105]. While both of these laws characterize word frequency data well, neither of the explanations that accompany them seem very realistic. In Zipf's 'Force of Unification', Zipf states that the speaker wants to minimize effort by reducing the lexicon size to one [111]. Clearly a lexicon of one is not very conducive to communication and it seems it would minimize effort as much as a human trying to communicate in binary. Mandelbrot's use of information theory starts with the set of words [112], but clearly a one word lexicon existed before a lexicon with two words. Therefore, Mandelbrot's reasoning totally eliminates the inclusion of an evolutionary process, which is core to the development of language [132] [133].

Zipf's Law does not hold for large corpuses. From Figure 4.1, there is a change in regime around the $10,000^{th}$ rank. There will be several plots like Figure 4.1 in this chapter and they all have some common features. As customary, all of the Zipf plots are drawn on a log-log scale. Zipf's Law was not fitted to the data, but the canonical form, where $\alpha = 1$, was included in all of the plots. Therefore, the line for Zipf's Law was drawn through the word of first rank and it has a slope of -1. As seen in Figure 4.1, there is a second regime of word usage. For all of the plots, the line for the second regime crosses the line for Zipf's Law at the $10,000^{th}$ rank and has a slope of -2 ($\alpha = 2$). There was no data fitting to keep the display of data consistent across plots. Moreover, the specific power law exponents are not of utmost important for this analysis.

Google removed any words from the data that did not appear in the corpus at least 40 times (from supplemental information in [70]). Since only part of the full Google Books

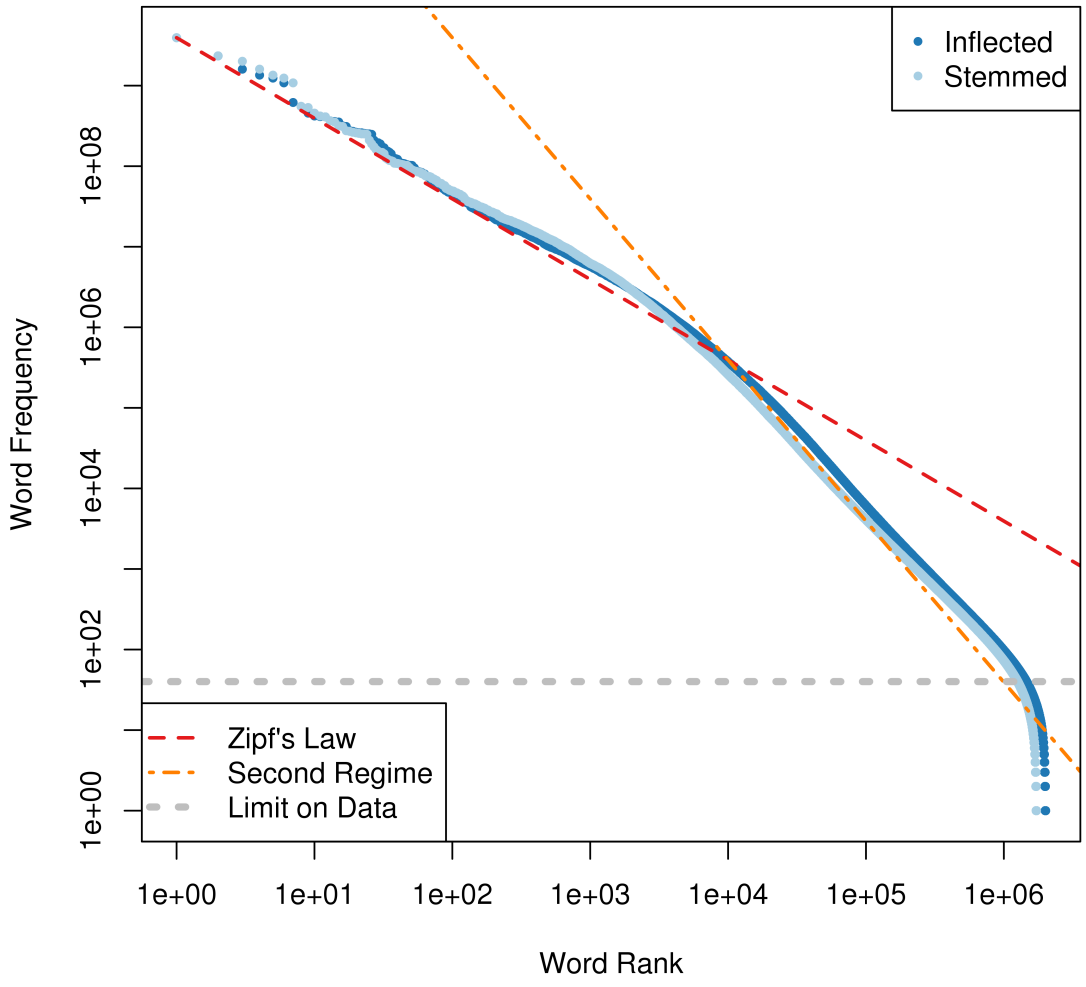
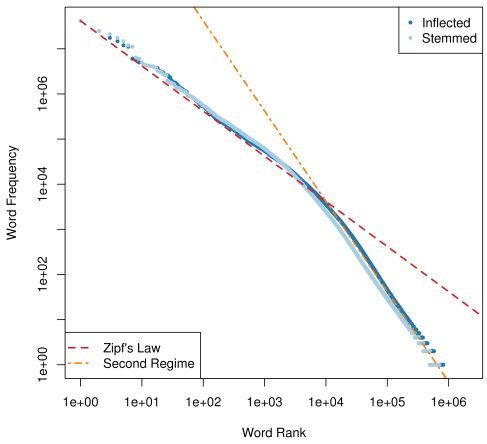


Figure 4.1: Zipf plot showing the word frequency.

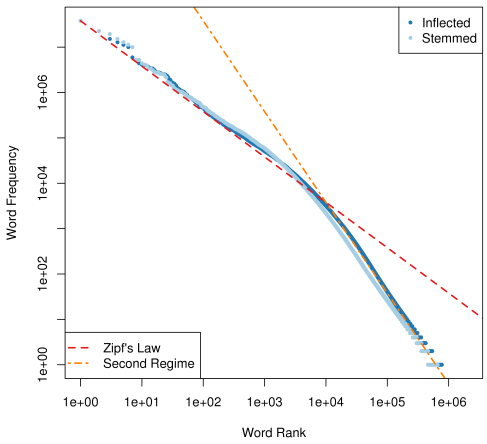
corpus is used here (from 1900 to 2000), it is possible to have words that appear fewer than 40 times, but the tail of the distribution is still cropped (e.g., a word that appears 39 times in a single year). The final line on all of the Zipf plots shows this limit on reliable distribution data.

The deviation of the most frequent word (ranks 1 through 7) from Zipf's Law is not the main concern here, in part because this topic has been explored extensively starting with Mandelbrot [112]. However, it is worth noting that normally the most frequent words appear less frequently then would be predicted by Zipf's Law and as discussed earlier, this is the reason for the better fit of the Zipf-Mandelbrot Law to empirical data. With this extremely large corpus, the opposite holds true, where the most frequent words appear more often than would be predicted. Tsonis and collaborators [134] found that the most frequent words appeared less often than expected and that if Zipf's Law held for those words then the language would be "rather awkward and possibly not an efficient means of communication" [pg. 13] [134]. However, this statement is not supported by the empirical data discussed here. Tsonis and collaborators [134] also suggested the deviation is due to the structural role of those words in the language. While I agree that these most frequent words have special structural significance, it does not appear that they necessarily need to appear less often then would be predicted by Zipf's Law. Moreover, a Zipf plot posted on the Internet [135] of a large corpus from Wikipedia also displays this same pattern where the first several ranks have a higher frequency than predicted by Zipf's Law. The Wikipedia data also shows the two regimes of word usage [135], which is the focus here.

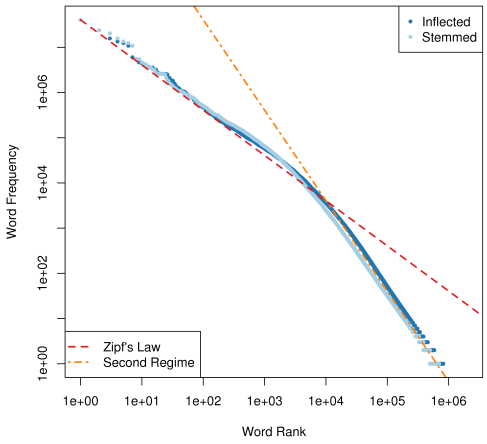
Word usage of two [104] [105] and three [136] [137] regimes have been recognized in other large bodies of text. Here I systemically examine subsets of the word frequency data to see what effect it has on the two regimes. For example, the Google Books data includes the year the words were used and Figure 4.2 shows six temporal snapshots (every 20 years from 1900 to 2000). The change in regime is a consistent pattern that has appeared over the last one hundred years. The systemic evaluation of subsets of words tests the persistence of the two regimes of word usage and has the additional benefit of evaluating the reliability of the



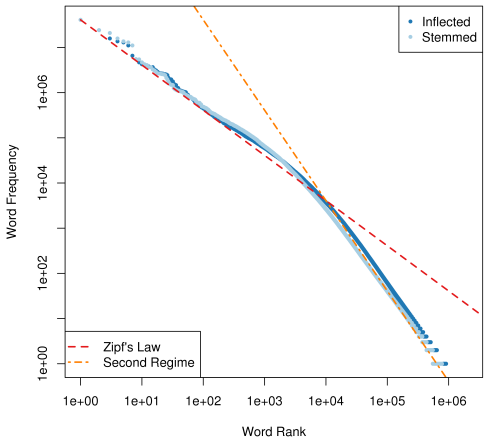
(a) 1900



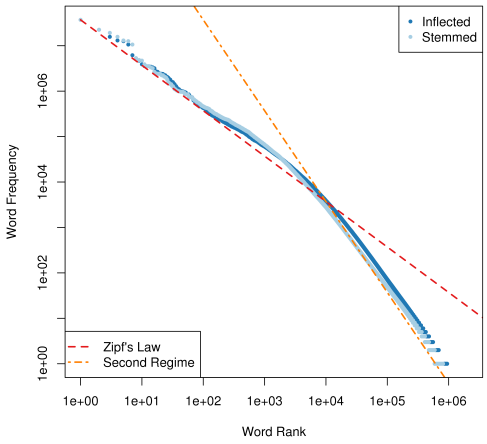
(b) 1920



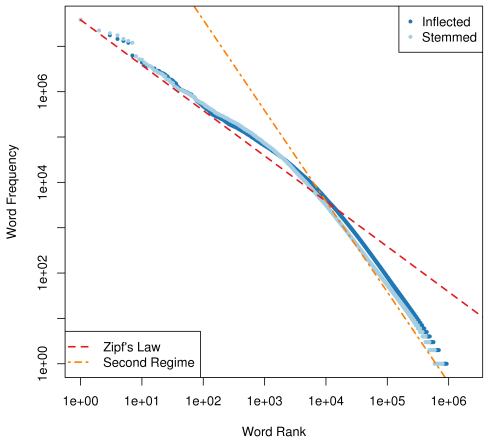
(c) 1940



(d) 1960



(e) 1980



(f) 2000

Figure 4.2: Zipf plot showing the word frequency by year.

Google Books data.

Ferrer i Cancho and Solé [104] have suggested that the first regime consists of versatile words that form the "kernel lexicon" and the unlimited second regime is used for "specific communication". Words found in a dictionary[6] form the core lexicon of a language. Therefore, if only dictionary words are used in the Zipf plot then much of the second regime should disappear. Figure 4.3 shows that much of the second regime did disappear, although there is still a regime change. To determine if the regime change occurred due to the same words being in the top ranks, the percentage agreement of the top ranks were calculated (see Appendix C.1). In all cases, the change in regime occurred without a major change in the words in the first regime.

Another way to examine the second regime is to look at only the words that appear every year for the entire 101 years. Naturally, this will cut off the tail of the distribution, but as seen in Figure 4.4 the regime change still occurred. Finally, the subset of dictionary words that appear in all 101 years were examined and the regime change still occurred (see Figure 4.5). The robustness of the regime change is surprising, especially when considering how many unique words are lost when only using dictionary words (see Table 4.2). These results show that the regime change was not generated by some random noise in the data, which is possible as discussed later in the next section.

4.2.1 Models of Word Use

There have been several attempts to model word usage. Even the very simple models, can produce similar results as empirical word usage data. For example, models have shown that random sequences of letters produce Zipf's Law [138] [139] [140] [141]. In these models, commonly referred to as "typing monkey" models, sequences of letters are generated at random. There is a probability of a space ($P(s)$) and the 26 letters occur uniformly random with probability $\frac{1-P(s)}{26}$. The word sequences created by this random process follow Zipf's

[6]Unix and similar operating systems (e.g. Mac OS X and linux) include a word list (located in /usr/share/dict/words). This word list is generally used for spell checking and other text processing tasks. The word list used for this analysis is from Fedora 16. The word list had 415,834 entries.

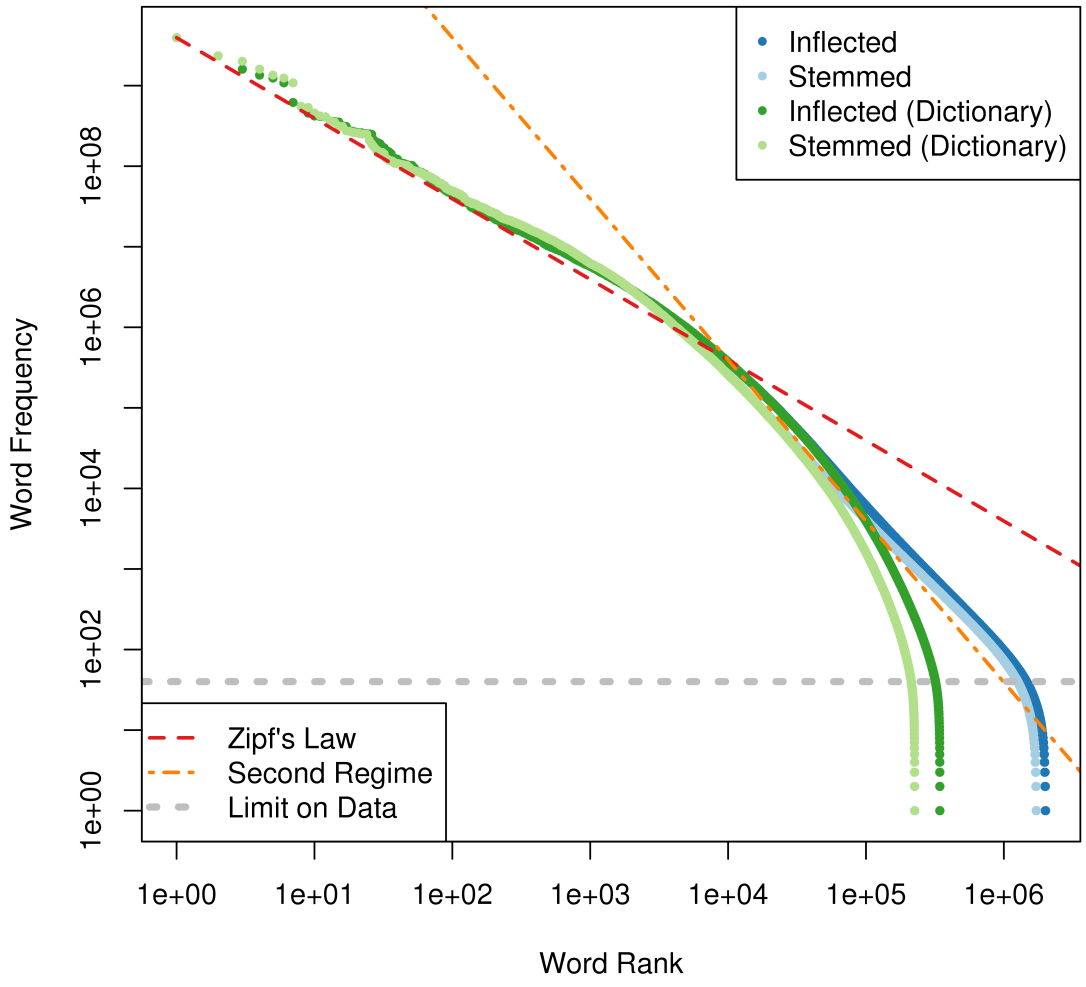


Figure 4.3: Zipf plot showing words that appear in a dictionary.

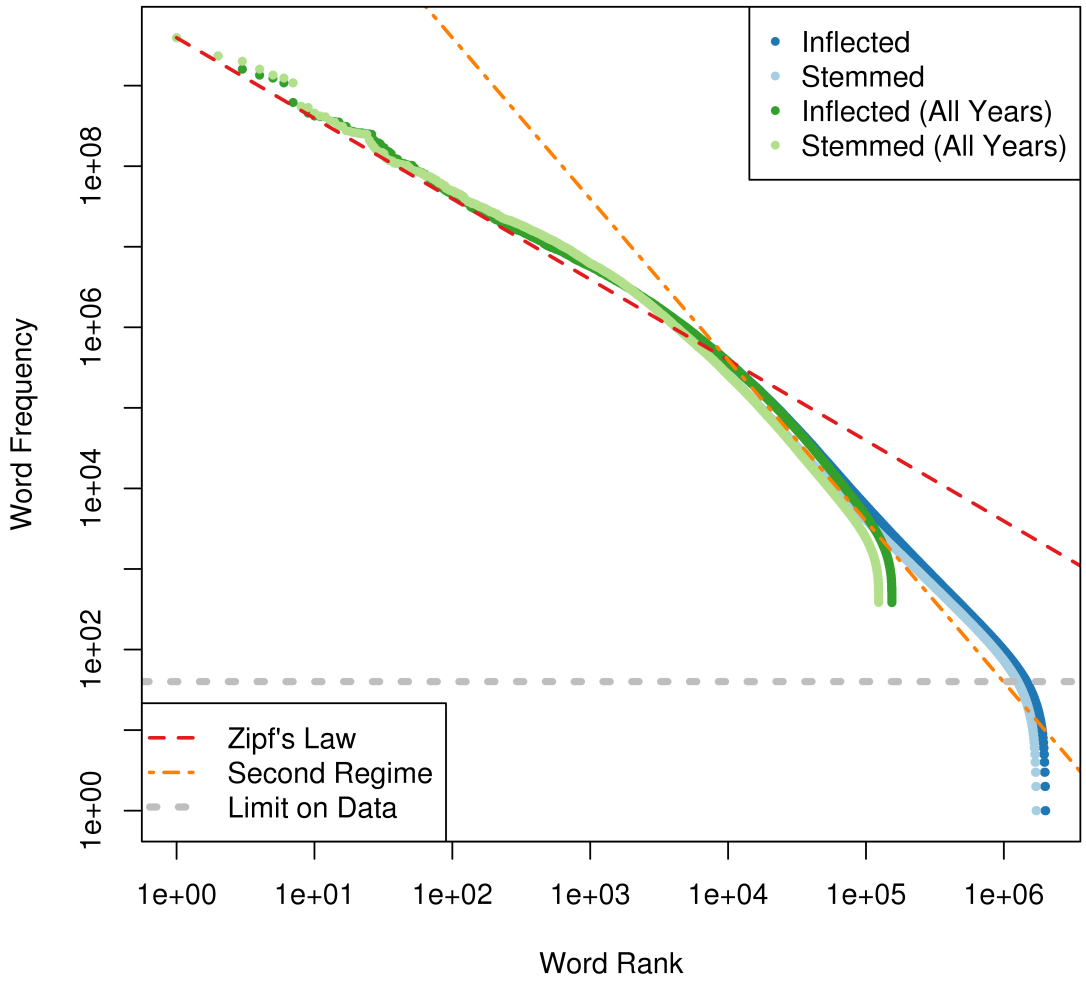


Figure 4.4: Zipf plot showing words that appear in all years (1900 to 2000) of the data.

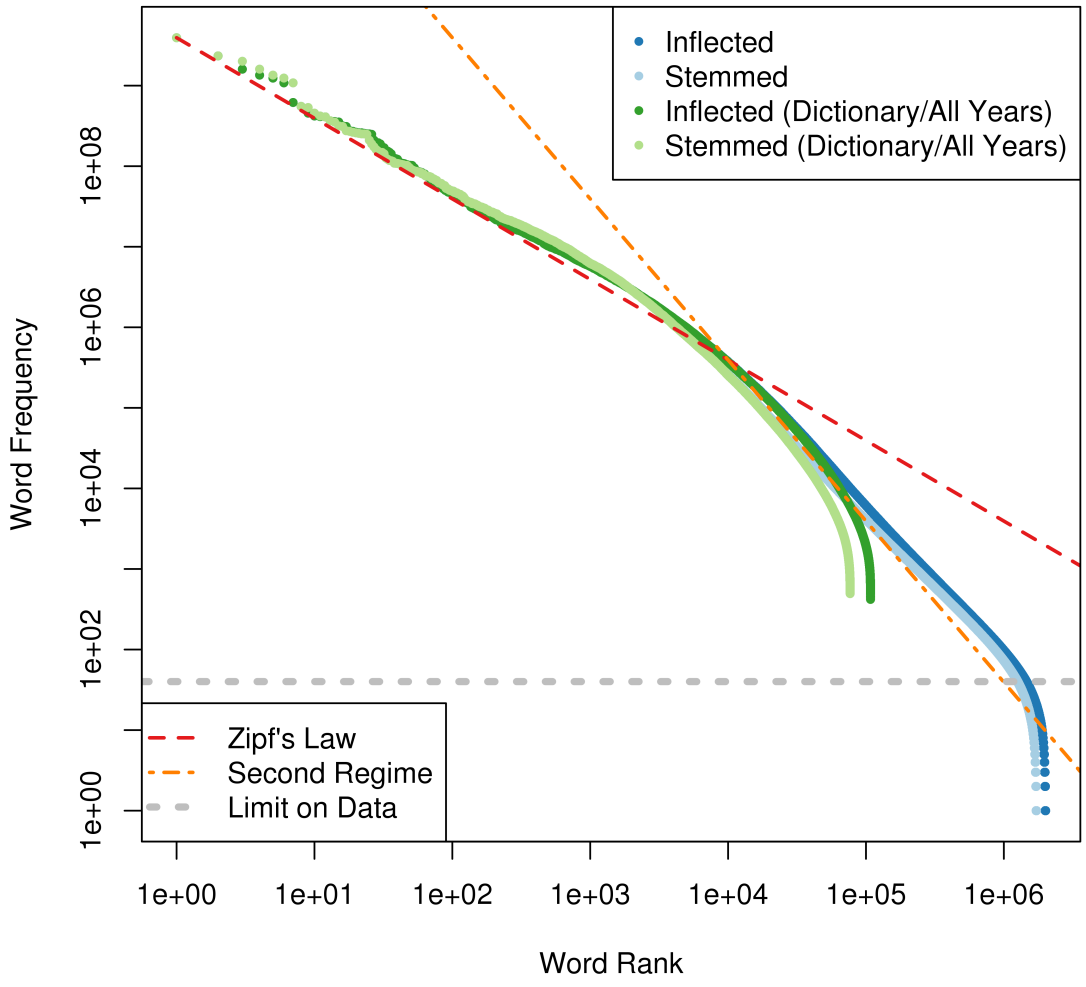


Figure 4.5: Zipf plot showing dictionary words that appear in all years of the data.

Table 4.2: Total word occurrences and total unique words for each of the versions of the word frequency data.

Data Set	Total Word Occurences	Total Unique Words
All Words		
Inflected	53,410,329,926	2,007,922
Stemmed	53,410,329,926	1,723,434
Dictionary Words		
Inflected	52,728,044,562	343,402
Stemmed	52,728,044,562	225,427
All Words - All Years		
Inflected	52,662,000,782	154,111
Stemmed	52,734,300,895	123,597
Dictionary Words - All Years		
Inflected	52,364,382,364	107,879
Stemmed	52,420,887,187	76,738

Law[7] [138] [141].

Google predicted a 98% accuracy rate for the word frequency data [70], although 2% of 53 billion word occurrences could generate lots of random words. By examining only the dictionary words and words that appear every year for a century, it shows that the second regime is not simply a transition from a first regime of purposeful text to a second regime of "typing monkeys". Therefore, the Google Books data seems reliable and the two regimes are a persistent feature.

Before moving on, a few more comments about models that generate random sequences of letters. While it produces Zipf's Law, there are other properties of these randomly generated sequences that deviate from human language. For example, the median rank (r_{median}) is much higher for the random sequences than for language [143]. The median rank is the rank where probability for ranks 1 to r_{median} sum to 0.5. For human written

[7]Perline later found that the random sequences follow a similar distribution, the log-normal distribution with a power law tail [142].

text, words of a given length follow Zipf's Law while random sequences do not [144]. Also, the distribution of word length is not the same in random sequences of letters (i.e., random language follows an exponential distribution for word length [144], but human language follows a log-normal [145] [146] or Poisson [147]). Finally, the vocabulary grows faster in random texts [148].

While the "typing monkeys" model recreated word use by constructing sequences of letters, the following model by Simon [62] directly used words as the building blocks to create artificial text. Simon's model [62] uses a Yule process [149]. In this model, at each step, a new symbol (word) is added to the end of a string of symbols (words). With probably α_s, the next symbol is a new one that has not appeared before.[8] Otherwise, the next symbol is randomly selected from the existing string of symbols. Simon found that this model generates a Zipfian distribution with slope of $\alpha_s - 1$ [62].[9] This is also similar to the preferential attachment process proposed by Barabási and Albert [150].

Gan and collaborators [151] found that Simon's model does not generate data that follows the n-tuple Zipf law, so they made a modification to the model. The n-tuple Zipf Law is similar to the original Zipfian relationship, but it is regarding sequences of n letters. It says that if you break the text up into sequences of length n, then those sequences follow Zipf's law [152][153]. If there are N letters (or symbols) in a piece of text, then there will be $N-n+1$ occurrences of a n-tuple. In the growth model proposed by Gan and collaborators [151], sequences of symbols are generated using a simple copy and paste operation. The model starts with an initial randomly generated string of symbols. In each iteration of the model, a random portion of the existing string is selected to be copied and pasted to the end of the string [151]. For their experiments, they chose an exponential distribution to determine the length of the sequence of symbols to be copied [151]. They found that their model recreated the n-tuple Zipf law that is found in real symbol sequences (e.g., DNA, language, computer codes[10]), but did not recreate the long range correlation found in the

[8]Note that this α_s is different then the α used in Zipf's Law (Equation 4.2).

[9]Therefore, $-(\alpha_s - 1)$ is α in Zipf's Law.

[10]The α values found in these different types of data may differ from the α of 1 that is commonly found

real data [151]. In short, the modification to Simon's model by Gan and collaborators [151] made it possible to include structure within the generated sequence of symbols.

Dahui and collaborators [155] attribute the emergence of Zipf's law as a consequence of the growth of vocabulary size and the preferential selection of words. Chinese characters do not follow Zipf's law for ranks greater than approximately 1,000 [156], while many other languages do (e.g. English, French and Spanish)[155]. Furthermore, the growth in the number of Chinese characters is much slower than words in English. With their model, they were able to recreate the frequency distribution of English words (Zipfian) and Chinese characters (not Zipfian) [155]. Chinese characters fall somewhere between English words and English characters (two Chinese characters is approximately one English word [156]) in terms of the role that they play in their respective language. The frequency usage of the 26 English letters do not follow Zipf's law [151]. The frequency of n-grams for Chinese (sequences of Chinese characters) and English (sequences of English words) both follow Zipf's Law [156]. This is evidence that the distribution of segments of the language's structure follows Zipf's Law.

Zanette and Montemurro [157] noted that Simon's model does not allow for faster decay of low frequency words (i.e., the second regime). They modified Simon's model in two linguistically reasonable ways [157]. First, they introduced sub-linear introduction of new words and secondly, made new words more likely to be reused. They found the modified model fits empirical data in finer detail [157].

The final two mathematical models of Zipfian behavior are used to fit the Google Books data to determine which model provides the better fit of the data. The models are promising because they both have the potential to fit the first and second regime of word usage. The first model was motivated by empirical word frequency data [104] and borrows a mathematical equation [105] that was originally used by Tsallis in folded protein research [158]. The second model uses an abstract model to develop a mathematical formulation that does not depend on the specifics of the system [159][160]. These two formulations represent a

in human language. For example, noncoding DNA was found to have an $\alpha \approx 0.3$ [154].

broad spectrum of models inspired by a range of empirical and theoretical motivations.

Montemurro's formulation [105] was based on empirical observations of large corpuses of text and is given by the equation[11]:

$$P(f) \propto \frac{1}{\mu f^r + (1-\mu) f^q} \tag{4.4}$$

Here $P(f)$ is the probability a word has frequency f. This formulation has two power law exponents, r and q, and the blend of these two parameters is controlled by the μ parameter. Since there are two power law exponents, this model will be referred to as the 'power-power' model.

Baek et al. [159] proposed the random group formation (RGF) model, which makes no assumptions about the underlying system other than that items are divided into groups. In the case of word usage, an item (ball) being placed in a group (slot) is analogous to a word being used, thereby the number of groups is the number of unique words [159]. The following distribution is the most likely one if you are placing balls in slots and there is equal chance of finding a specific ball in a specific slot [159][12]:

$$P(f) \propto \frac{1}{\exp(bf) f^\gamma} \tag{4.5}$$

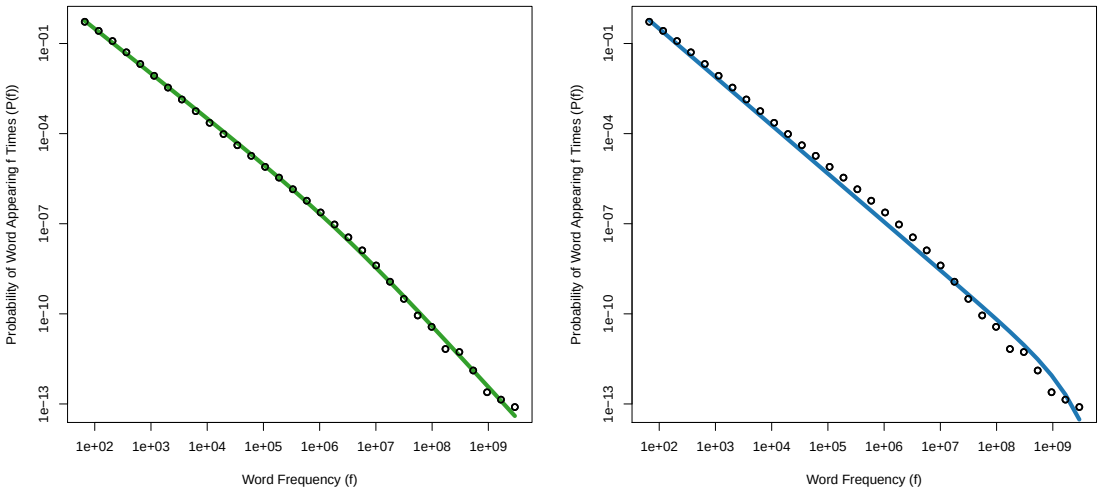
The $P(f)$ is the same as before. The γ parameter is the exponent for the power law and the b parameter is the rate for the exponential. This is a power law with an exponential cutoff, so it will be referred to as the 'power-exponential' model.

The parameters for both equations, 'power-power' and 'power-exponential', were fitted

[11] In Montemurro's paper [105], the equation had another term. The original equation was: $P(f) \propto \frac{1}{\mu f^r + (\lambda - \mu) f^q}$, but for the purposes here the λ term was not needed and was replaced by 1. Also, without the extra term, convergence is easier because there are only two fits that are best (i.e., changing μ to $1-\mu$ would swap the role of r and q and thereby, create two best fits).

[12] Other models have suggested a first regime characterized by a power law and a second regime by an exponential. For example, in their continuum theory, Albert and Barabási's modified preferential attachment model produces this type of two regimes in network degree distribution [161].

to the empirical data using the Gauss-Newton algorithm[13]. The probability density function of the data was computed using logarithmic binning to smooth the data (32 bins were used). During fitting the word frequencies (f) were normalized so they would sum to one. To avoid fitting the higher probabilities in the density function at the expense of the lower probabilities, the log of the probability density function was fitted. Four versions of the data was used in the fitting procedure. In additional to fitting all of the data, the subset of words from the dictionary was also fitted. Each of these sets of data had corresponding inflected and stemmed versions, which created the four sets of data.

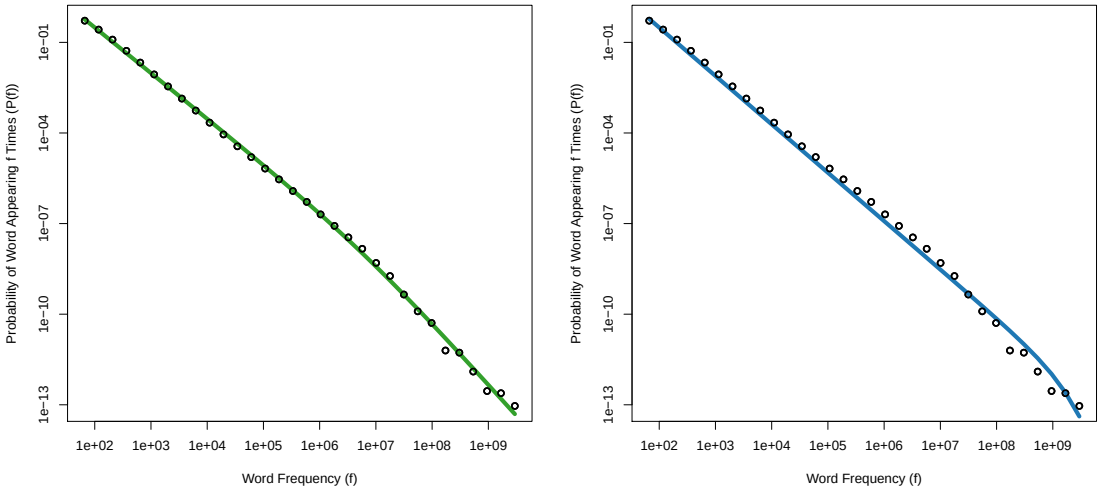


(a) power-power (b) power-exponential

Figure 4.6: Fit of all inflected words to each of the models.

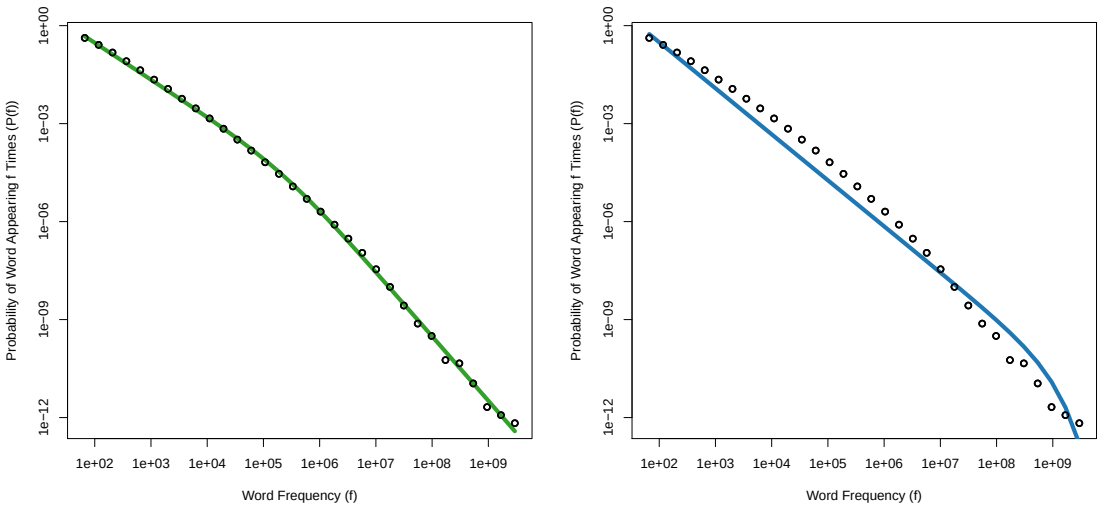
Figures 4.6 through 4.9 show that both models appear to provide a good fit of the

[13]The Gauss-Newton algorithm is a method for solving nonlinear least squares problems. The method's ability to converge is dependent on providing a good first guess of the parameters. Therefore, the adaptive nonlinear least squares algorithm (from the 'port' library [162]) was used to generate the starting point for the parameters. Also, maximum likelihood estimation was used to ensure that the generated parameter estimates were valid. Those estimates are very similar to the ones presented here and can be found in Appendix C.2.



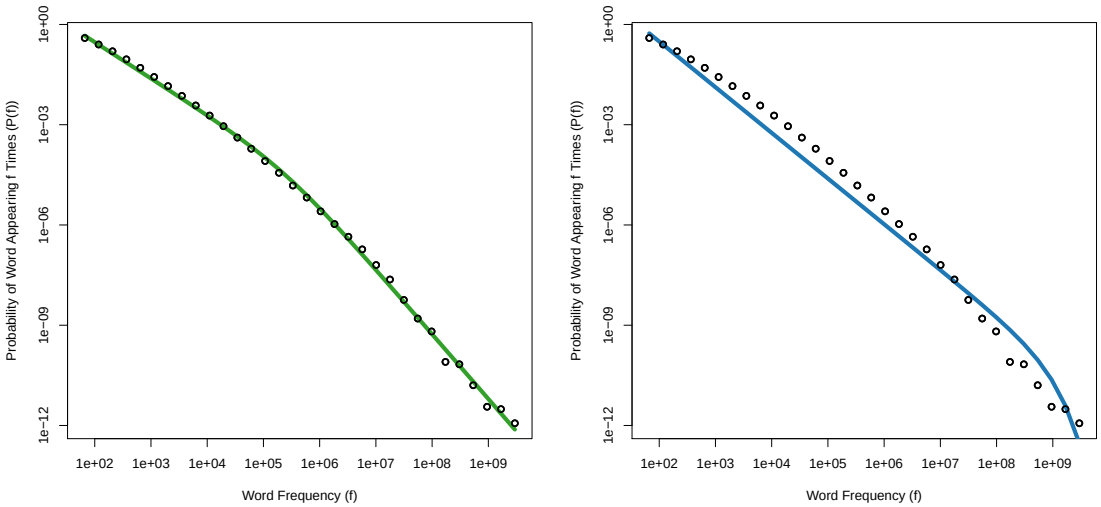
(a) power-power (b) power-exponential

Figure 4.7: Fit of all stemmed words to each of the models.



(a) power-power (b) power-exponential

Figure 4.8: Fit of inflected dictionary words to each of the models.



(a) power-power

(b) power-exponential

Figure 4.9: Fit of stemmed dictionary words to each of the models.

empirical data. Table 4.3 provides a summary of the parameters with the best fit. For both models, the parameter estimates for the data with all inflected words (not just the dictionary words) were on the same order as empirical data used in their original paper. In Montemurro's paper [105], for a corpus of 2,606 books, parameter estimates[14] were $\lambda = 0.86$, $\mu = 0.0022$, $q = 1.95$, and $r = 1.32$. Baek et al. [159] analyzed two corpuses: novels by Thomas Hardy and novels by Herman Melville. The estimate for $\gamma = 1.656$ for Hardy's novels and $\gamma = 1.734$ for Melville's novels [159][15]. It should be noted that the data sets used here are considerably larger[16] than either of the data sets used by Montemurro [105] or Baek et al. [159].

Two metrics were used to measure fit quality of each model: the correlation (R) and the KS-statistic (D). The correlation (R) is the Pearson's correlation coefficient between the

[14]To convert from the version of the power-power equation with λ [105] and the one without it, use the equation: $\mu_{new} = \frac{\mu_{old}}{\lambda}$.

[15]The estimated values for b do not appear to be reported in Baek et al.'s paper [159].

[16]The largest data set used here is 291 times larger than Montemurro's data [105] and 39,791 and 71,820 times larger than the Hardy and Melville data sets used by Baek et al. [159], respectively. See Table 4.2 for more information about the data used here.

Table 4.3: Parameter estimates for both models (power-power and power-exponential).

	power-power parameters			power-exponential parameters	
Data Set	q	r	μ	γ	b
All Words					
Inflected	2.06	1.49	0.01	1.61	5.30
Stemmed	2.07	1.51	0.02	1.60	4.80
Dictionary					
Inflected	1.98	1.13	0.0002	1.41	9.38
Stemmed	1.94	1.08	0.0002	1.37	9.61

empirical data's probability density function and the model predictions. The KS-statistic (D) measures the maximum distance between the cumulative density function of the empirical data and the model predictions. Therefore, higher correlation values (R) and lower KS-statistic values (D) indicate a better fit. For all four versions of the data set and for both metrics, the power-power model provided a better fit of the data than the power-exponential model.

The fit quality listed in Table 4.4 does not necessarily give a definitive answer to which model is best. For example, the power-power model has one more parameter than the power-exponential model, which adds to the model complexity. The Akaike information criteria (AIC) [163] allows for the comparison of the two models while considering the differences in number of independent variables. The AIC is expressed by [163]:

$$AIC = -2 \log L + 2k \tag{4.6}$$

The L variable is the likelihood of the model parameters given the empirical data, which is equal to the probability of the empirical data given the model parameters. The k parameter is the number of independent variables in the model. Smaller AIC values indicate a better

Table 4.4: Fit quality of each model (power-power and power-exponential).

	power-power fit quality		power-exponential fit quality	
Data Set	R	D	R	D
All Words				
Inflected	.998	.041	.995	.069
Stemmed	.997	.052	.994	.074
Dictionary				
Inflected	.996	.052	.982	.129
Stemmed	.993	.067	.975	.146

Note:
$-1 \leq R \leq 1$ and larger values are better.
$0 \leq D \leq 1$ and smaller values are better.

model, so each independent variable in the model introduces a penalty to the AIC. Another information criteria metric that imposes a larger penalty for each additional independent variable is the Bayesian information criteria (BIC) [164], which is defined by:

$$BIC = -2 \log L + k \log n \tag{4.7}$$

The L and k values are the same as before and the n parameter is the number of observations.

Table 4.5 displays the AIC and BIC for both models, and both metrics show that the power-power model is the better model for all four versions of the data. In addition, the quality of fit was better for the power-power model (see Table 4.4). Since the power-power model shows more promise, the two regime hypothesis is supported by the empirical data more than the RGF model. There appears to be another mechanism at work here other than randomly placing balls in slots. The next section will discuss this mechanism and how it relates to the hierarchical structure of language.

Table 4.5: Model comparison using AIC and BIC of the two models (power-power and power-exponential).

	power-power information criterion		power-exponential information criterion	
Data Set	*AIC*	*BIC*	*AIC*	*BIC*
All Words				
Inflected	7.1	13.0	66.0	70.4
Stemmed	16.5	22.3	61.2	65.6
Dictionary				
Inflected	2.6	8.5	102.2	106.6
Stemmed	18.3	24.1	103.2	107.6

Note:
For AIC and BIC, smaller values are better.

4.2.2 Importance of Hierarchical Structure

In the previous analysis, the model with two power law regimes provided the best fit with the empirical data. This section explores the hierarchical structure of language and the important role it plays in the two regimes of word usage.

Hierarchical structure in language has been recognized as an important consideration since the early days of Zipf's Law when Mandelbrot studied it in the 1950s [165] [166]. Models that have indicated the importance of the structure of language include Kanter and Kessler's work [167], which put forward a Markov process[17] that could recreate Zipf's Law. Their transition matrix included structure so it would not create totally random sequences of words [167]. With their Markov process, they also were able to recreate a deviation from Zipf's Law in low frequency words, although their decay was exponential [167] (and not the power law decay that was found here).

The structure of the relationships between words have also been explored. For example,

[17] A Markov process is a stochastic process where the next state is only dependent on the current state and a transition matrix.

Ferrer i Cancho and Solé [168] created a word network where the nodes represented words and two words were linked if they appeared in the corpus next to one another. The two regimes of word usage were evident in the degree distribution of the word network [168]. They found the word network was a small world network [169], which means that there was a large clustering coefficient and short average path lengths. The node degree distribution was also a power law with a similar exponent as the networks generated by the preferential attachment model [150] , which means that the word network's degree distribution is similar to that found in the Internet [127] [128]. Ferrer i Cancho and Solé [168] proposed that as the word network evolved, newly introduced words were more likely to attach to existing words that were highly connected.

Dorogovtsev and Mendes [170] used a modified version of the preferential attachment mechanism [150] to recreate the word network found in the empirical data by Ferrer i Cancho and Solé [168]. In standard preferential attachment [150], at each time step, a new node is attached to an existing node with probability proportional to the existing node's degree. In this modified version of preferential attachment [161] [171], at each time step, a new node was added as before, but additional edges were also added to the network with probability proportional to the product of the degrees of each pair of nodes. This method was able to reconstruct the degree distribution with two regimes as seen in the empirical word network [170].

This type of word network is of interest from a complex systems perspective, but does not represent the type of hierarchical structure that is most critical to the underlying mechanism behind the two regimes of word use. The hierarchical structure of language is lost due to the way the word network flattens the word sequences that are collocated. If the ability to collocate was paramount, then the word 'a', which can replace 'the' and still create a syntactically correct English sentence, could appear in a corpus as the most frequent word, which to my knowledge[18] has never happened with a corpus of substantial size[19]. The

[18]Mandelbrot [166] noted that the word 'I' can sometimes be the most frequent word, but he does not give an indication of the size of the corpus. Regardless, the point still stands because 'I' is not a syntactically correct replacement for 'the'.

[19]From the data used here, the word 'the' was the most frequently used word for every year for the entire

word 'the' is the most common word in English not because it can be collocated with many different words, it is collocated with many different words because it plays a structural role in the English language. It is this structural role that leads to 'the' being the most frequently used word. In short, a word's versatility is due to its structural role and not vice versa.

Naturally, the structure of language has been studied extensively in linguistics. Chomsky and collaborators [172] [173] suggested the X-bar theory where individuals build sentences using a hierarchy. A sentence is composed of a noun phrase (NP) and a verb phrase (VP), and the NP and VP are composed of subcomponents, which in turn can be made of subcomponents, thereby creating a hierarchy [172] [173]. Chomsky also proposed that varying forms of the same semantic content that use similar words, called 'surface forms', can be traced back to a common structure, called 'deep structure' [174]. Finally, Chomsky's Universal Grammar puts forward the idea that all languages are governed by universal rules [175] [176], which gives some explanation of why Zipf's Law is so universal in human languages.

Simon has noted the importance of hierarchy in complex systems [29] and language is a complex system [177]. Hierarchical systems that are nearly decomposable are more easily evolvable than systems where all of the components are interconnected [29]. Pinker [178] states that while language is communicated as a sequence of words, it is constructed in a modular way, where sentences are created using a 'phrase structure tree'. The modularity of language allows different phrases to be interchanged between sentences [178].

When characterizing the two regimes of language, the first regime is marked with versatility and structure, while the second regime is typified by idiosyncratic phrases [104] [105]. However, empirical evidence supporting this distinction is lacking. Here I present support for the importance of hierarchical structure in differentiating the first and second regime. For the Google Books data, each word was tagged with its part-of-speech using the

twentieth century.

Stanford log-linear part-of-speech tagger[20] [179] [180]. The top 50 ranks[21] for each of the parts-of-speech are listed in Appendix C.3.

Figures 4.10 and 4.11 display the Zipf plots with each of the parts-of-speech plotted separately. Two extra lines are included in these Zipf plots. There is a vertical line at the $10,000^{th}$ rank and a horizontal line that passes through the intersection point of the vertical line and the two Zipf's Law lines (first and second regime). The grey points in the figures include all of the data, so each data point in the parts-of-speech data series move to the right to be aggregated into the grey data series.

From these figures, it is clear that the majority of structural words (referred to as function words or closed-class words) are located in the first regime. These parts-of-speech include the conjunctions, prepositions, determiners, pronouns, and WH words (e.g., which, when, who, what). Some verbs should probably be placed in this structural category, including auxiliary verbs (e.g., is, was, be, have, were) and modal verbs (e.g., will, would, can, should, could), because they generally carry little semantic content. Moreover, these verbs appear quite frequently, and thus are in the first regime. The second regime is almost entirely composed of non-structural words (referred to as content words or open-class words). These parts-of-speech include nouns, verbs, adjectives, and adverbs. These figures illustrate that once the distribution no longer contains elements required to create a hierarchical structure, there is a regime change.

[20]The Stanford NLP libraries can be found here: http://nlp.stanford.edu/software/
[21]Not all parts-of-speech include 50 words.

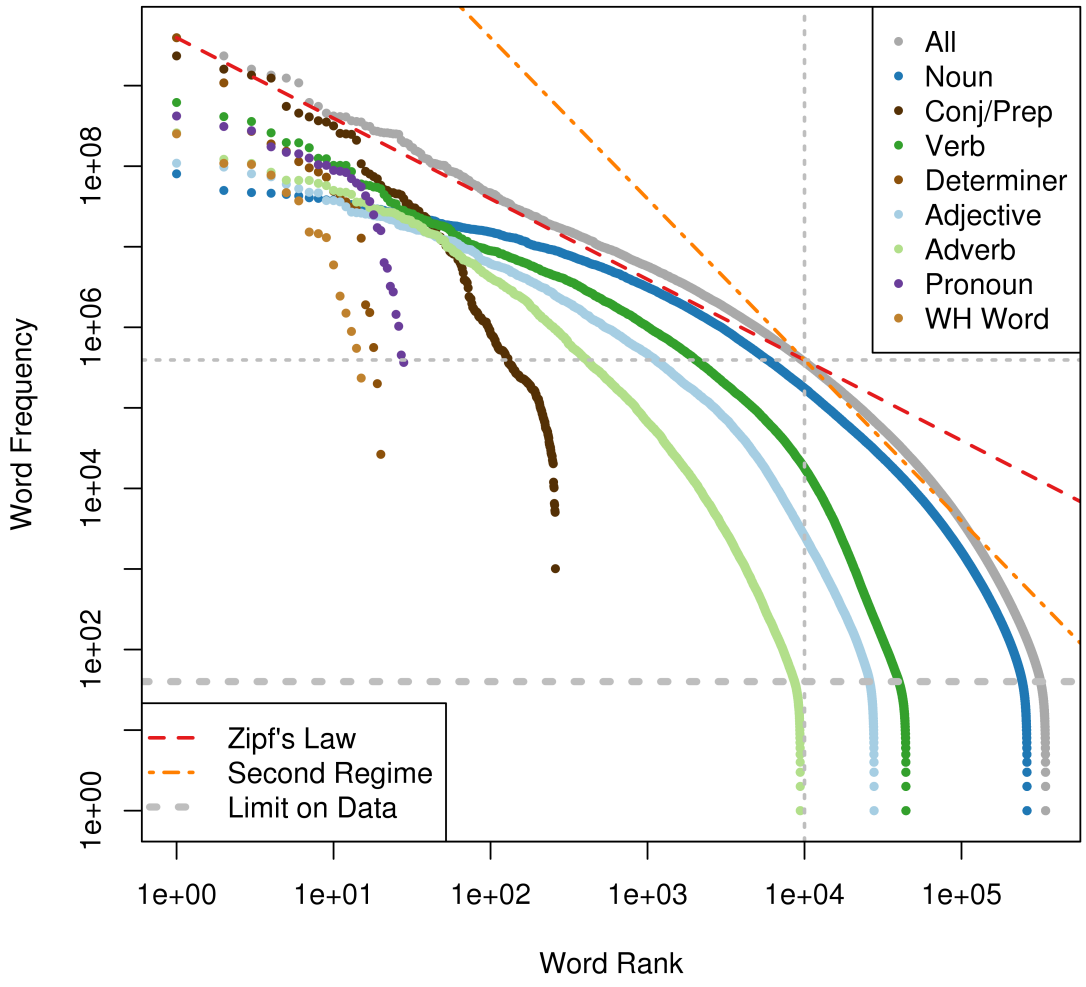


Figure 4.10: Zipf plot showing words (inflected) that appear in a dictionary categorized by part-of-speech.

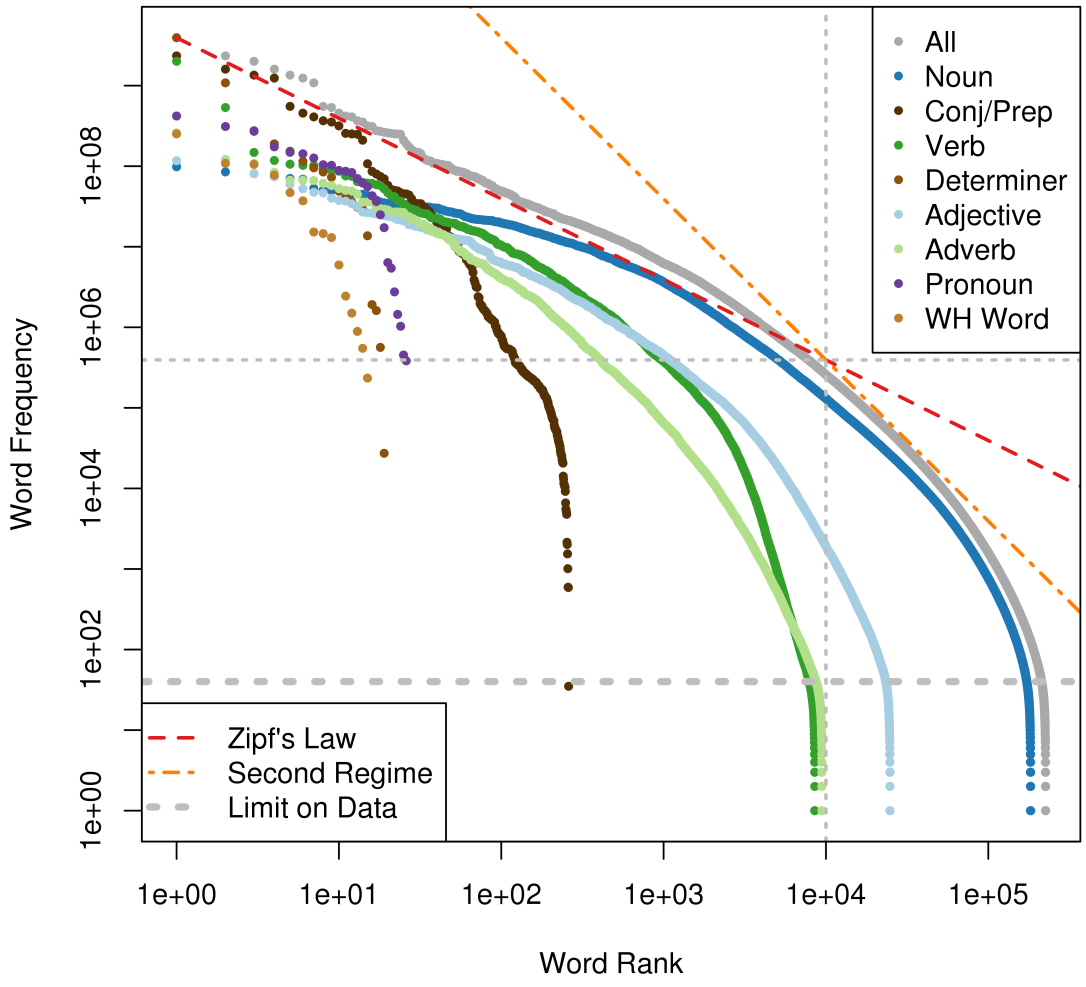


Figure 4.11: Zipf plot showing words (stemmed) that appear in a dictionary categorized by part-of-speech.

4.3 Analysis of Change in Word Usage

In this section, the change in word use over time is the focus of the analysis. Figures 4.12 and 4.13 provide an illustration of word usage dynamics over the 101 years of Google Books data. For both the inflected and stemmed words[22], the top 50 ranked words for the entire 101 years were calculated and their ranks for each year of the 101 year period are displayed. On the left and right side of the figures, the words are listed in order of their overall rank for the entire 101 years. These figures clearly show how the most frequent words consistently remain at the top, while as word frequency decreases there are many more changes in rank over time. While this is not that surprising considering the differences in word usage is much greater for ranks i and $i+1$ than for ranks $i+1000$ and $i+1001$, it provides intuition for the remainder of this analysis.

4.3.1 Empirical Growth Rates

The change in word usage can be measured by a growth function defined by:

$$g = \log \frac{S_{i+1}}{S_i} \tag{4.8}$$

Where S_i is the frequency of a word at time i. The S_i values were normalized by dividing by the total number of word occurrences for year i to avoid confounding results due to unequal numbers of words for each year. This equation allows for symmetrical growth rates (i.e., $\log \frac{a}{b} = -\log \frac{b}{a}$), and thus the distribution of growth rates are centered approximately at zero.

Figure 4.14 shows the growth rate distributions for each of four data sets (i.e., all inflected, all stemmed, dictionary inflected, and dictionary stemmed). Since the y-axis is on a logarithmic scale, a tent shaped distribution indicates a double exponential distribution, which is also referred to as a Laplace distribution. The figure shows a 'nested' Laplace

[22]For Figures 4.12 and 4.13, there is no difference between the dictionary words and all words.

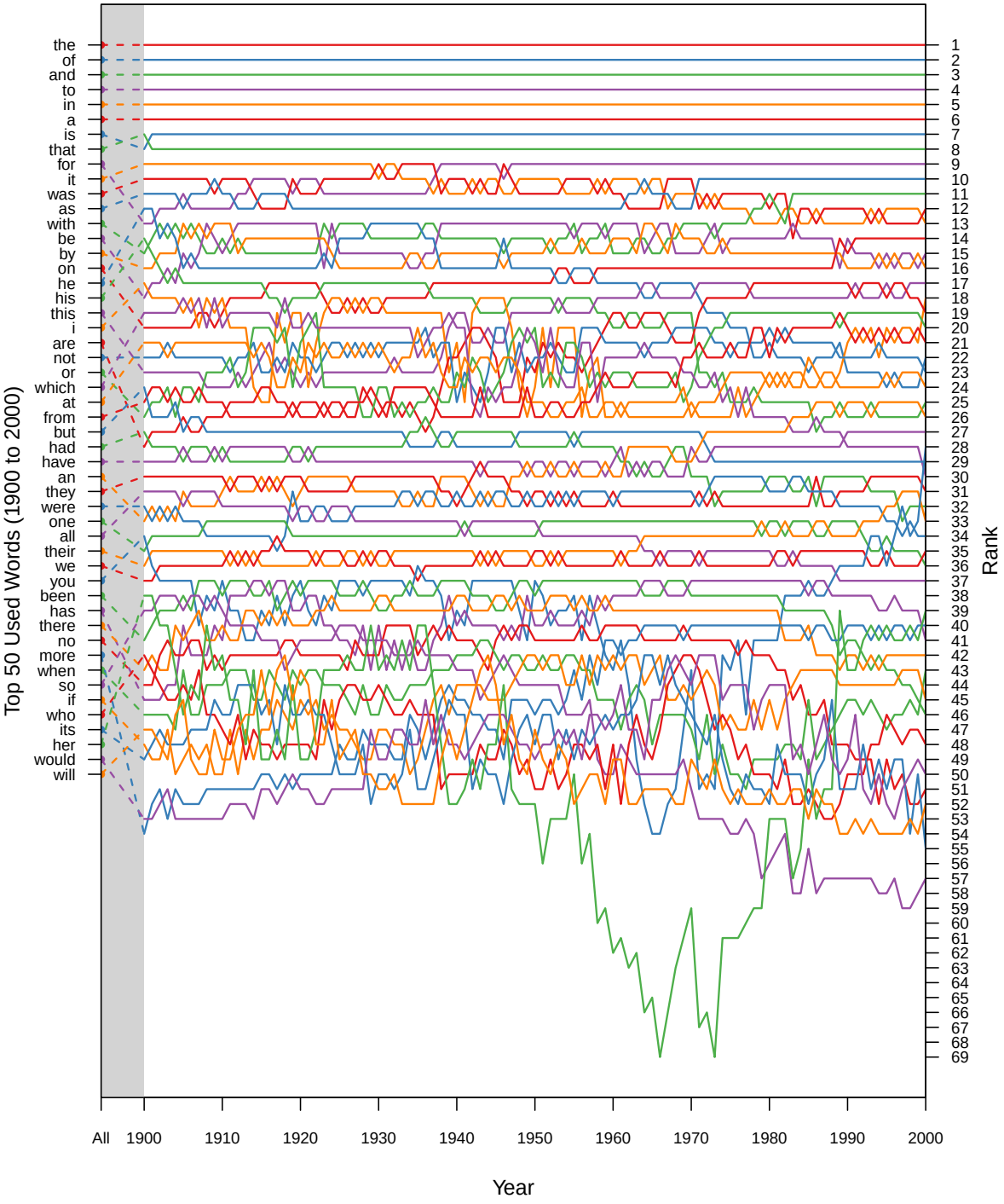


Figure 4.12: Changes in rank of usage over 100 years for the top 50 inflected words.

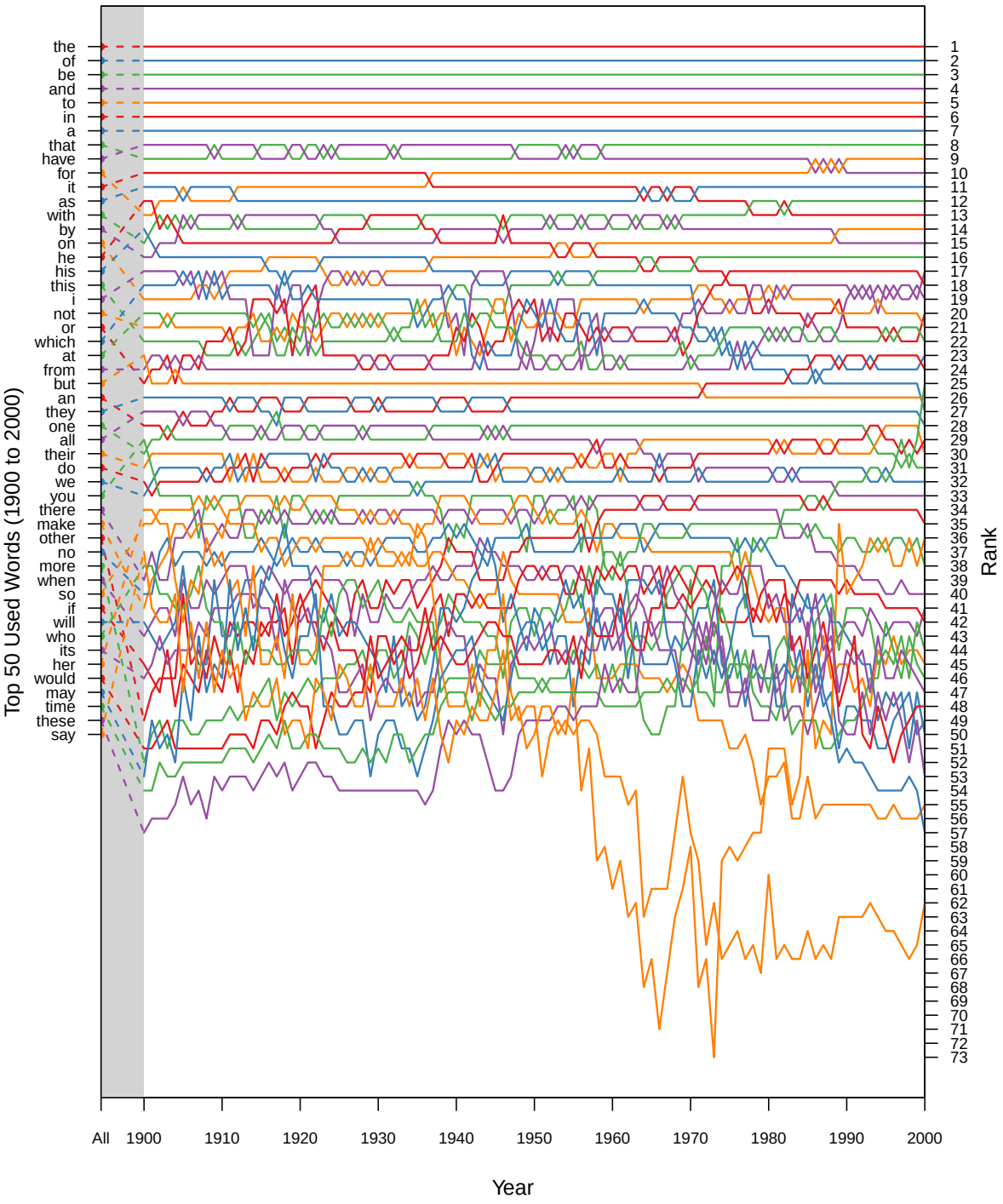


Figure 4.13: Changes in rank of usage over 100 years for the top 50 stemmed words.

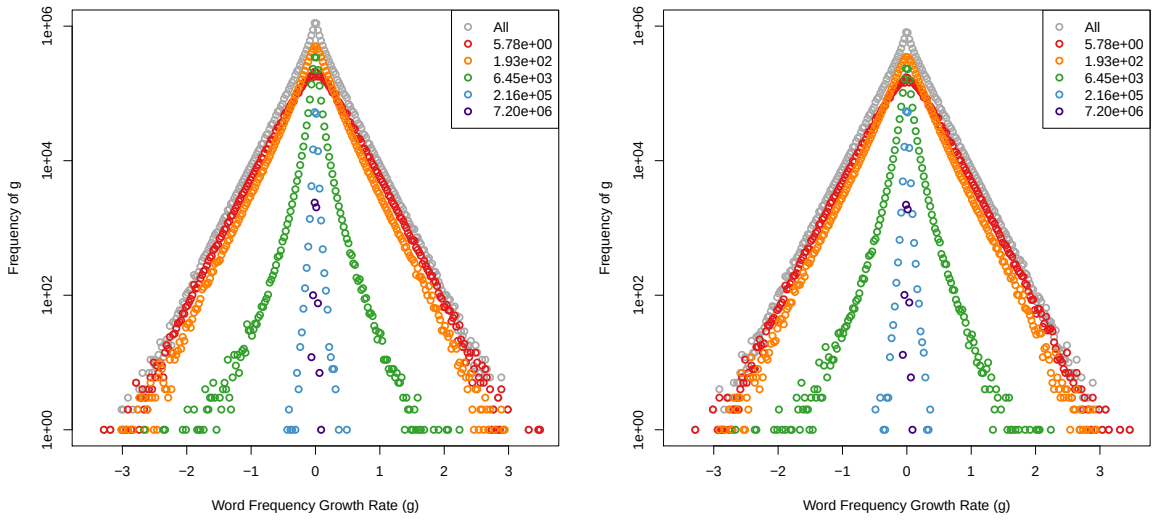
distribution for the word usage growth rates as evident by the tent shaped distributions that are embedded inside one another. The nesting occurs due to higher frequency words having a smaller variance in growth rates compared to lower frequency words. The data set was divided into five bins using logarithmic binning based on the usage of the word in the first year (1900). The geometric mean[23] of each of the bins are indicated in the plots. Stanley et al. [106] found this same 'nested' Laplace distribution when analyzing the growth rates of firms with regards to the amount of sales and number of employees. Moreover, other empirical firm data suggested that the growth rates of firms follow a Laplace in the middle range of growth rates, but have power law tails with an exponent of 3 [181]. However, the word frequency data does not support this hypothesis because the tails of the distribution drop more quickly than a power law with an exponent of 3.

Another interesting point in Figure 4.14 is that the words in the middle bin (i.e., geometric mean $= 6.45 \times 10^3$) have fatter tails than would be expected. This middle bin is centered approximately at the transition between the first and second regimes, so perhaps this could account for some of the extra weight in the tails. Nevertheless, all five bins are tented shaped as well as the distribution of all words grouped together, thusly the growth rate of words follow the 'nested' Laplace pattern. It is also worth noting that if the number of occurrences of a word drops to zero for a year, then the growth rate for that year and the next is undefined[24], thereby those values were not included in the growth distribution.

Stanley et al. [106] also found that the change in standard deviation over initial value dropped with a power law relationship. Figure 4.15 shows the relationship between initial value and standard deviation of the growth rate. The initial values are the word frequencies at the first year (1900) and those frequencies were logarithmically binned. For each of these bins, the standard deviation was calculated for the growth rates for the entire 101 years. Therefore, each word could have up to 100 growth rates if the word appeared in the corpus every year for the entire 101 years. For the smallest initial values, a boundary condition of zero occurrences limits the amount of variance possible in those ranks causing a flatting

[23] Geometric mean$(x) = \exp\left(mean\left(\log\left(x\right)\right)\right)$

[24] This due to the divide by zero or the log of zero in Equation 4.8.



(a) All Inflected Words

(b) All Stemmed Words

(c) Dictionary Inflected Words

(d) Dictionary Stemmed Words

Figure 4.14: These figures display histograms of the growth rate for each of the four data sets. The 'nested' Laplace growth distribution is present in each of the versions of the data.

of the curve. For the largest initial values, there is a break in the power law relationship at approximately rank 7. It is also worth noting that from looking at the changes in rank over time in Figures 4.12 and 4.13, the first 6 inflected words and first 7 stemmed words do not change rank during the entire 101 year period. With the additional evidence from Figure 4.15, it appears that these most frequent words play a distinctive role in the English language. Overall a power law relationship is reasonable for all but the lowest frequency words.

Figure 4.15 also displays slopes of two lines for reference. The first line with slope of $-\frac{1}{2}$ is what would be expected if the central limit theorem governed the growth process where the aggregation of a greater number of independent 'shocks' cause a decrease in variability [106] [182]. The second line with slope of $-\frac{1}{6}$ is approximately the slope found in the firm data by Stanley et al. [106] and falls within the range of values found in other empirical firm data (ranged from 0.14 to 0.2) [183]. Using results from firm size data, Riccanboni et al. [183] suggested that the slope of the relationship changed from zero for firms of size zero to $-\frac{1}{2}$ as the firm size approached infinity. This could also be true for the word frequency data, although the flattening out of the relationship for words of lower frequency may simply be caused be the boundary condition where it is impossible to have negative occurrences of a word.

To gain a better intuition of the relationship between standard deviation and initial value, it is easier to discuss it in terms of firms. Stanley et al. [106] proposed that if the firm's subunits were being strictly managed from the top down, then all subunits of the firm would behave in a similar manner, thereby there would not be a reduction in variance of firm growth as the number of subunits increased. Essentially, in this scenario of "absolute control", the subunits of the firm could not dampen the amount of variance in the firm's growth because the subunits could not behave independently thereby, there would not be a decrease in the growth rate's variance as the firm's size increased (i.e., the slope of the relationship would be zero) [106]. Conversely, if all subunits of the firm were behaving independently, then the slope of the power law relationship would be $-\frac{1}{2}$ due to the central

limit theorem [106]. Since the drop in variation for the word frequency data is flatter than the line with slope of $-\frac{1}{2}$, it shows that the word frequency data, like the firm data, does not follow a standard Gaussian growth process.

Continuing to follow the analysis of firm data from Stanley et al. [106], they found that after rescaling the 'nested' Laplace distribution that it collapsed to a single curve. The following equations were used to rescale the probably of the growth rates and the growth rates[25] [106]:

$$p_{scal} = \sqrt{2}\,(\sigma(g_{s_0}))\,p(g_{s_0}) \tag{4.9}$$

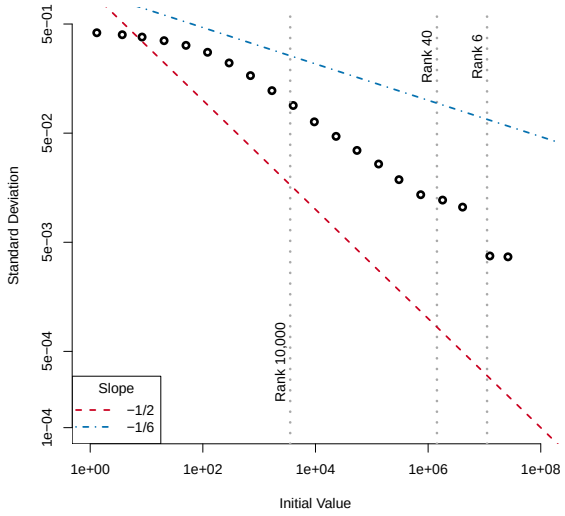
$$g_{scal} = \sqrt{2}\frac{(g - \overline{g_{s_0}})}{\sigma(g_{s_0})} \tag{4.10}$$

Figure 4.16 shows how the 'nested' Laplace collapsed to a single Laplace distribution. This exercise shows how the growth rates of word frequencies parallels the growth rates of firms. Recently, the Laplace distribution for word frequency growth was discovered for several languages (i.e. English, Spanish, and Hebrew) [184], but here I reconfirm this finding and establish the 'nested' Laplace as a more accurate characterization of word frequency growth. Furthermore, in the next section, a simple model is introduced that continues to explore this pattern of growth rates.

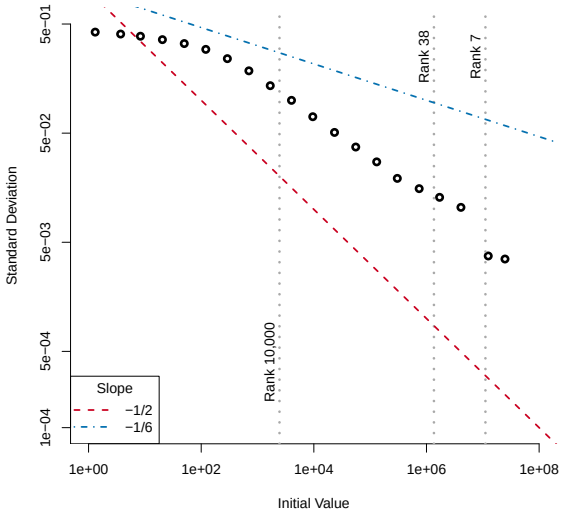
4.3.2 Explanation of 'Nested' Laplace Growth

When examining their firm data, Stanley et al. [106] proposed that firms' hierarchical structure created the 'nested' Laplace growth rates. The subunits of a firm are not acting in a totally independent or totally correlated way, but somewhere between, where policies implemented at the root of the firm's hierarchy can be modified as they transverse the branches of the organization [106]. Since larger firms generally have more subunits and

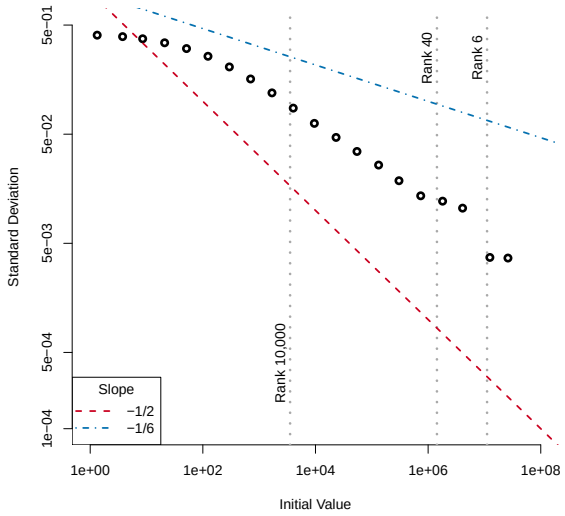
[25] I believe these equations are the same as Stanley et al. [106], but the notation has been changed. In Stanley et al. [106], the equation was written with $\sigma(s_0)$ instead of $\sigma(g_{s_0})$, but it seems that $\sigma(g_{s_0})$ is what was intended. If $\sigma(s_0)$ was indeed what was intended, then the normalization factor in Equation 4.10 would be the standard deviation of the log of the initial size, instead of what was used here, the standard deviation of the growth rate being rescaled.



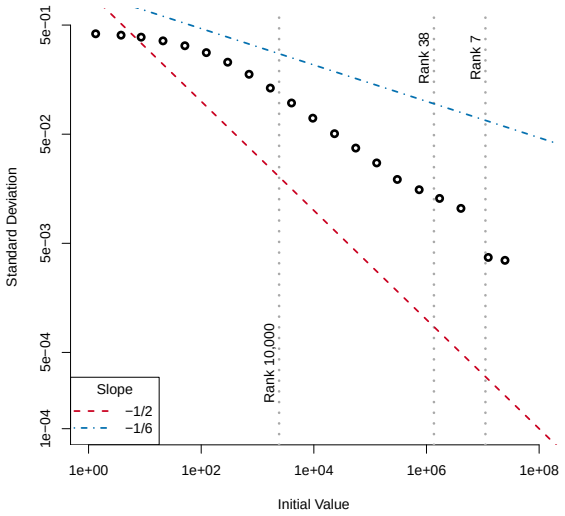
(a) All Inflected Words



(b) All Stemmed Words

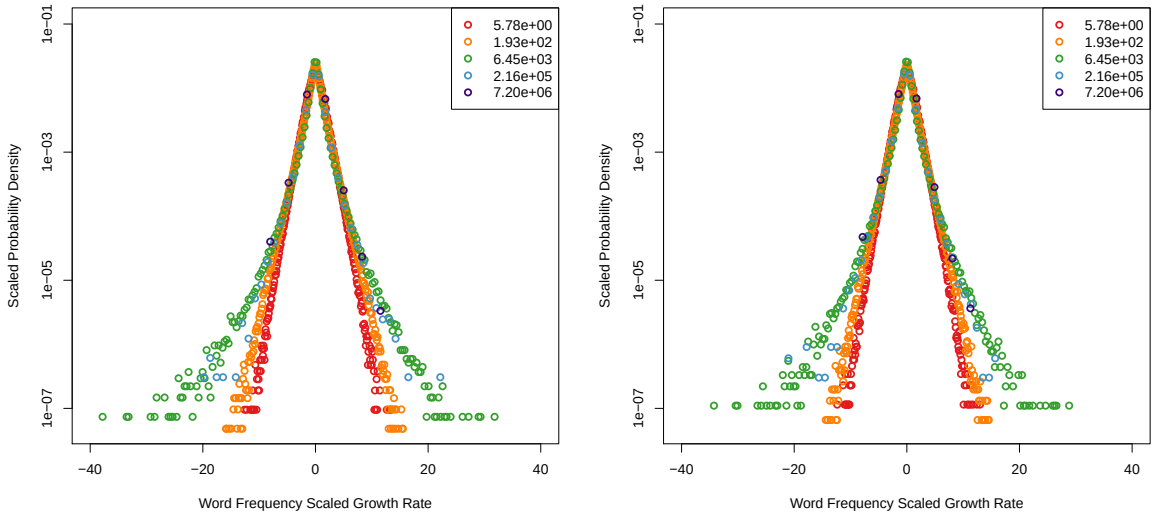


(c) Dictionary Inflected Words



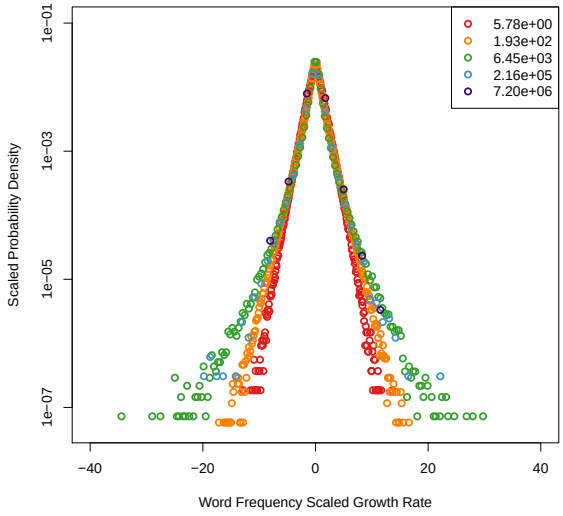
(d) Dictionary Stemmed Words

Figure 4.15: These plots show the relationship between initial value and standard deviation. For the smallest initial values, the power law relationship breaks due to the boundary condition of zero word occurrences. For the largest initial values, there is a break in the power law relationship, but overall a power law relationship seems reasonable.

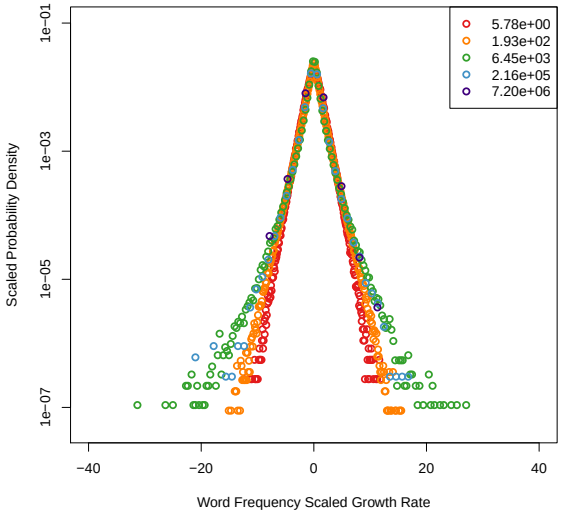


(a) All Inflected Words

(b) All Stemmed Words



(c) Dictionary Inflected Words



(d) Dictionary Stemmed Words

Figure 4.16: These figures display histograms of the scaled growth rate for each of the four data sets. This shows how the 'nested' Laplace growth distributions can be rescaled to collapse on one another. The center of the distribution rescales nicely, but the tails of the distributions do not rescale perfectly.

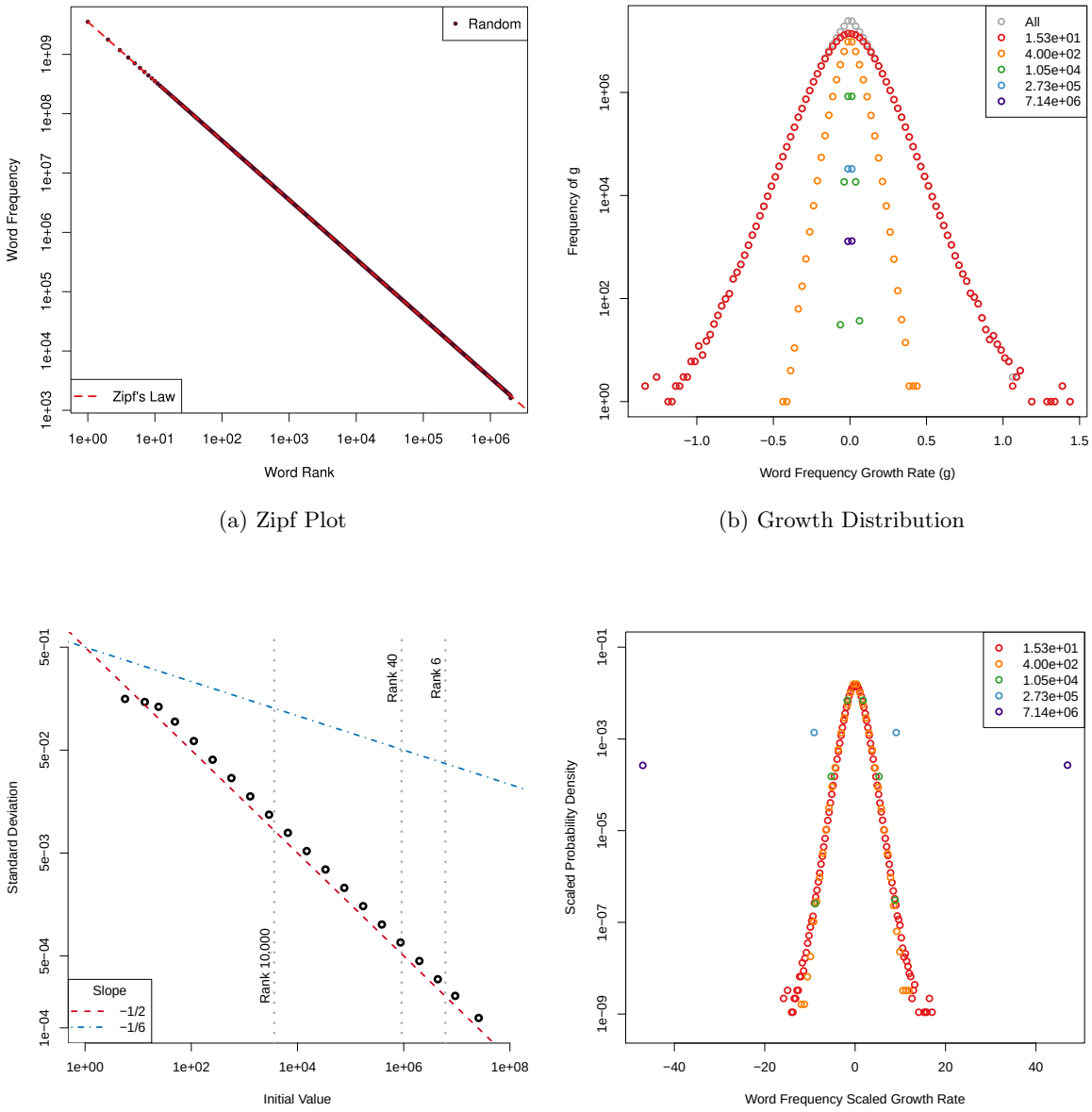
their behaviors are not totally correlated, the subunits' results can cancel one another and lessen the likelihood of an extreme overall growth rate.

This structural argument can also hold for growth rates of word usage. However, since both firms [122] and words [111] follow a Zipfian distribution, the pattern of growth rates could simply be a product of both types of data (i.e., firms and words) being drawn from similar underlying distributions. This hypothesis is tested using a simple null model. A null model tests what observables are caused by factors that are not of interest to the experimenter [185]. They are similar in nature to the zero-intelligence models studied by Gode and Sunder [59] [60]. Basically, a null model attempts to see what behavioral patterns naturally occur without any sophisticated mechanism where the agents behave randomly.

In this null model, a data set was generated using a Zipfian distribution for 101 years just like in the empirical data. The number of unique words and word occurrences from the largest data set (all inflected words) was used to parameterize the Zipfian distribution (see Table 4.2). Note that this Zipfian distribution is the basic Zipfian distribution and not the two regime power law that was discussed earlier in this chapter. The growth rates were calculated for the random data to see if it exhibits the 'nested' Laplace growth rates.

Naturally, the data fits Zipf's Law almost perfectly (see Figure 4.17a). From Figure 4.17b, the 'nested' distribution is visible. However, the growth rates have much less variance than the empirical data (see Figures 4.17b and 4.17c). From Figure 4.17c, there is a clear power law relationship between initial size and standard deviation, except for the lowest frequency words, which has the boundary condition of zero occurrences. However, the slope of this relationship is $-\frac{1}{2}$, thereby providing evidence that this growth process is governed by the central limit theorem and is of a different character than the empirical data discussed earlier. The growth distribution rescales almost perfectly, except for the two bins with the highest frequency words, which have only a few words and extremely small standard deviations that could be causing the rescaling to be sensitive to small changes in values.

Overall, the null model does a reasonable job of recreating the 'nested' Laplace growth rates and the power law relationship between initial size and standard deviation. However,



(a) Zipf Plot

(b) Growth Distribution

(c) Initial Value vs. Standard Deviation

(d) Scaled Growth Distribution

Figure 4.17: Results from the null model where data was generated using a Zipfian distribution.

the range of magnitudes of the growth rates are approximately half of that seen in the empirical data. There are no self reinforcing tendencies in the null model, but they are common in the empirical data (e.g., if a subject is mentioned once in a book, then it is more likely to be mentioned again). In summary, the null model performed surprisingly well considering its simplicity, thereby providing support that these patterns are a product of the same mechanism that created the Zipfian distribution. The presence of the Zipfian distribution is the only commonality between the three sets of data discussed here: firm data [122], word frequency data [111], and the data generated by the null model.

4.4 Discussion

In this chapter, subsets of the Google Books data were visualized to ensure the data provided reliable counts of word usage. The two regimes of Zipf's Law was a persistent feature in all versions of the data. After establishing the reliability and consistency of the two regimes, the fit of two mathematical models to the empirical word frequency data were compared. The random group formation (RGF) [159] is a mathematical formulation based on a simple abstract model with only one assumption about the process, that elements are divided into groups. However, the two regime formulation [105] that was motivated by large sets of empirical word frequency data [104] was found to be a better fit of the Google Books data. The Google Books data was tagged with parts-of-speech to show that structural elements of language were concentrated in the first regime, while the second regime was mainly content words. This provided support that the regime change was due to a structural shift of language from the first to the second regime. A hierarchical structure argument has also been proposed for explaining why city size distributions follow Zipf's Law [186] [187].

In the second analysis, the growth rate of the Google Books data was examined. Here a 'nested' Laplace (or double exponential) distributions of growth rates were found. Using the same analysis steps as Stanley et al. [106], the word usage data followed a similar pattern as the firm data. Both of these domains, linguistic and economic, have an underlying

hierarchical structure that is important for the development of these growth rate patterns (i.e. the 'nested' Laplace growth and the power law relationship between initial value and standard deviation).

Word usage [111] and firms [122] both follow a Zipfian distribution. To test if the 'nested' Laplace growth was simply random fluctuations produced by a Zipfian distribution, an artificial set of word frequency data was generated using a Zipfian distribution. The resulting data set displayed nesting and the power law relationship between initial value and standard deviation, but had less variance in the usage of words and a steeper slope for the power law relationship than the empirical data. Also, the ability of the growth rates to collapse on one another was a persistent feature in the null model, except for the most frequency words, which were very sensitive to small changes in standard deviation. Therefore, the ability to collapse the 'nested' distribution doesn't appear to be an indicator of a higher order innovation process. While the performance of the null model is somewhat mixed, overall it performed well considering its simplicity. Therefore, the same mechanism that created the Zipfian distribution may also be responsible for the patterns in growth rates.

In this chapter, I presented empirical evidence for the importance of a hierarchical structure when using building blocks. In the next chapter, an agent-based model of innovation that explicitly includes a hierarchical structure is introduced. This model is general enough to apply to many different types of innovation processes, including the modeling of word usage.

Chapter 5: Models of Dynamic Fitness Landscapes

This chapter continues investigating the material of the last chapter, but instead of merely describing the data, I now move toward explaining why the data have the structure they do. I do this by building an agent-based model that recreates the patterns from the empirical analyses from the previous chapter. The chapter begins with a description of this simple agent-based model and shows how it reproduces some of the stylized behaviors found in innovation research. The chapter concludes with a discussion of how the agent-based model is used to model word usage. The agent-based model is abstract enough to be applied to a variety of domains where building blocks are assembled into hierarchies. This chapter shows how a mechanism can be hypothesized using an agent-based model and then tested to determine if the model reproduces results similar to those found in the empirical data.

5.1 Agent-Based Model Description

In this agent-based model, there is social interaction between heterogeneous agents.[1] The model is co-evolutionary since agents create new goods that can change the agents' perception of their existing goods (i.e., a new good that is perceived as having higher fitness can cause an existing good to no longer be used). Over time, there is growth in complexity of new innovations, which consist of a growing number of components. This complexity is organized in a hierarchical structure. This model demonstrates creative destruction and punctuated equilibria.

[1] A earlier version of this work was presented at the second World Congress on Social Simulation [188]. I would like to thank the participants of this conference for useful feedback.

5.1.1 Model Setup

In this model, there are A agents and each of those agents can carry G economic goods. At the start of the simulation, N economic goods are generated and each good is assigned a random fitness, with a uniform probability distribution $U[0, 1]$. The initial set of goods is then randomly assigned to the agents. The agents independently evaluate the good and assign a perceived fitness to it. An agent's assessment of the perceived fitness of a particular good is random between 0 and the true fitness of the good. During each time step of the simulation, each agent has a small probability of combining two goods in its holdings and creating a new innovation. This new innovation will have a fitness randomly assigned between 0 and the sum of the two fitness values of the component parts. Each agent will then evaluate the new innovation and assign it a perceived fitness randomly between 0 and the fitness of the good. If the perceived fitness of the new innovation is higher than the lowest perceived fitness in the agent's current holdings, then the new innovation is added to the agent's holdings and drops the good with the lowest perceived fitness.

In this model of technological evolution, economic goods have a finite lifetime. An economic good's chance of survival is influenced by its fitness and the agents' perceptions of the good. Agent perception of a good is based on the idea that not all individuals will recognize the utility of a new innovation and furthermore, not all individuals will find the innovation equally useful.

5.1.2 Model Visualization

In additional to the modeling activities, this chapter also introduces visualizations for understanding the dynamics of the model. These visualizations include the standard histograms and line graphs, but also include domain specific visualizations where detailed evolutionary dynamics can be explored.

Time moves from left to right in Figure 5.1, and by virtue of the monotonic relation between time and utility, fitness also increases to the right. A tick mark is placed on the time axis each time the fitness achieved doubles. The irregular spacing of the tick

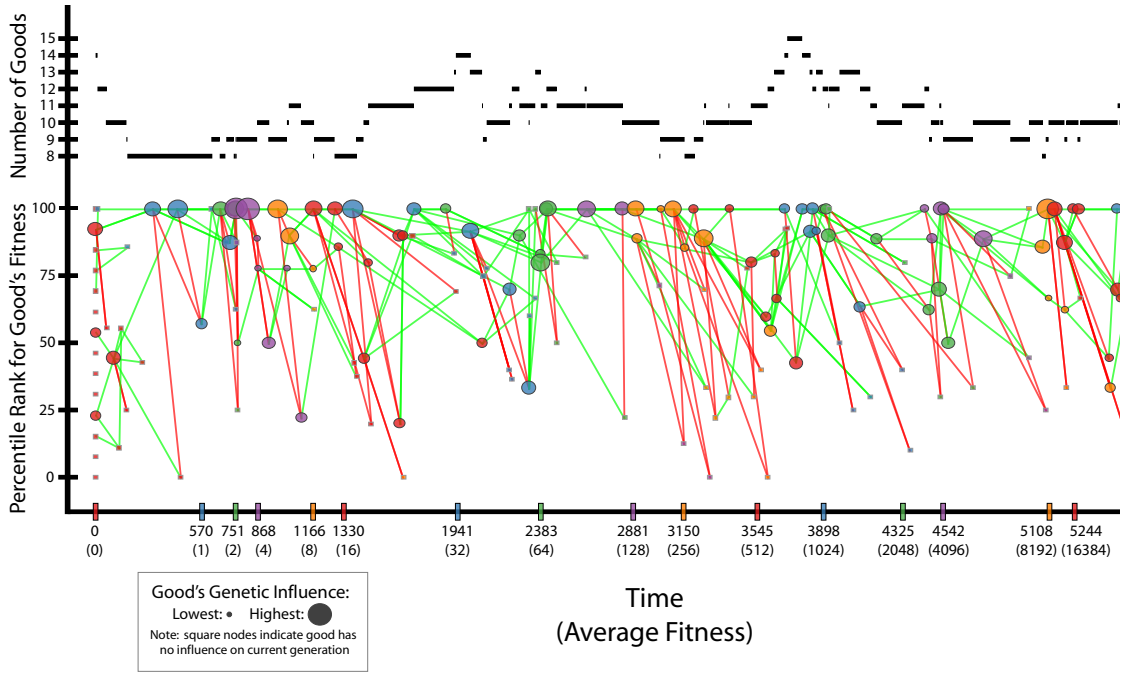


Figure 5.1: Visualization of the dynamics of the agent-based model. For this simulation run, there were 4 agents (A) that could hold up to 5 goods (G) and 20 initial goods (N).

marks illustrates irregularities present in the exponentially growing fitness. The size of a node reflects how many agents have adopted it. New goods usually have two technological antecedents, but occasionally only one since two of the same item can be combined to produce a new good. The colors group goods into fitness bins and each consecutive bin doubles in size. Since the fitness of the goods is monotonically increasing, the five colors simply cycle over time. Superimposed at the top of the figure is an indication of the number of goods present in the economy over time. When there is a sharp drop in the number of goods in the simulation, the cause of the drop can be explored in the bottom part of the figure, thereby providing understanding of the process of creative destruction.

This type of visualization can be used to illustrate different simulation setups. As a simple example, Figure 5.2 is the same type of visualization, but now the agents are separated into two teams or islands. There is a clear separation between the fitness values

achieved by one team compared to the other.

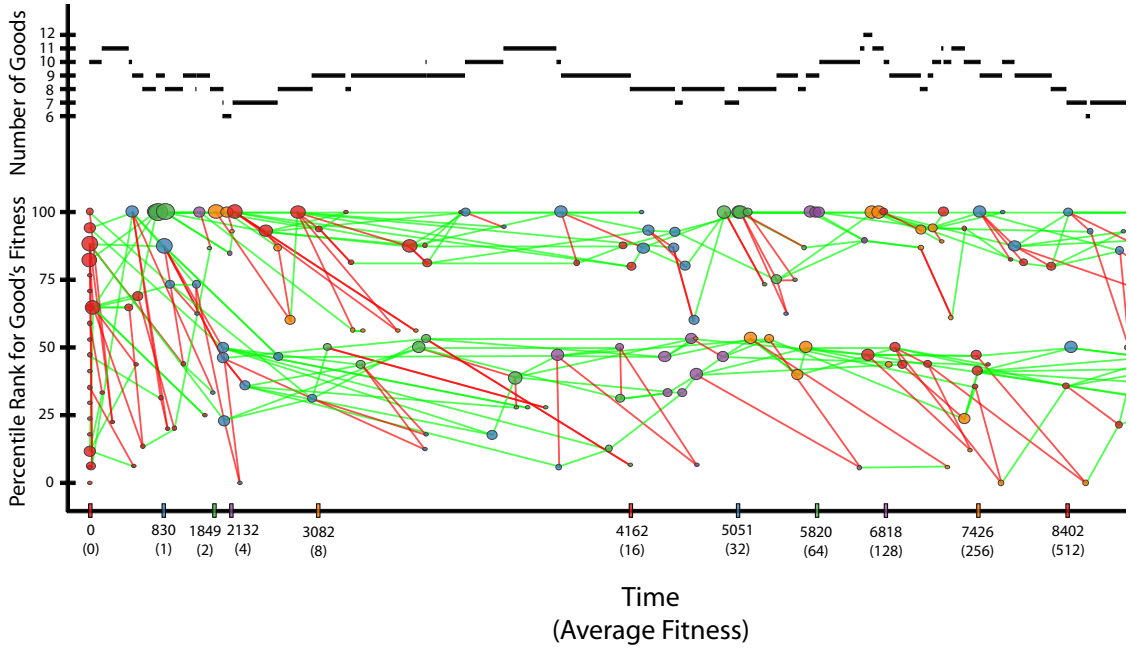


Figure 5.2: Visualization of the dynamics of the agent-based model with teams. This simulation run had the same parameter settings as the previous one ($A = 4$, $G = 5$, and $N = 20$).

5.1.3 Model Results

Figure 5.3 shows the irregular exponential growth in the average fitness of the agents in the simulation. The plot's y-axis is on a log scale, so the linear increase in the plot line shows an exponential growth in average fitness. Relatively stable periods followed by rapid growth are indicators of punctuated equilibria. Figure 5.3 also shows how the complexity of the goods grow over time. The figure shows an example good from early in the simulation and an example good late in the simulation run. The plot also shows how the x-axis from Figure 5.1 relates to a standard plot. The tick marks are close together during periods of

rapid grow and further apart during periods of relative stability.

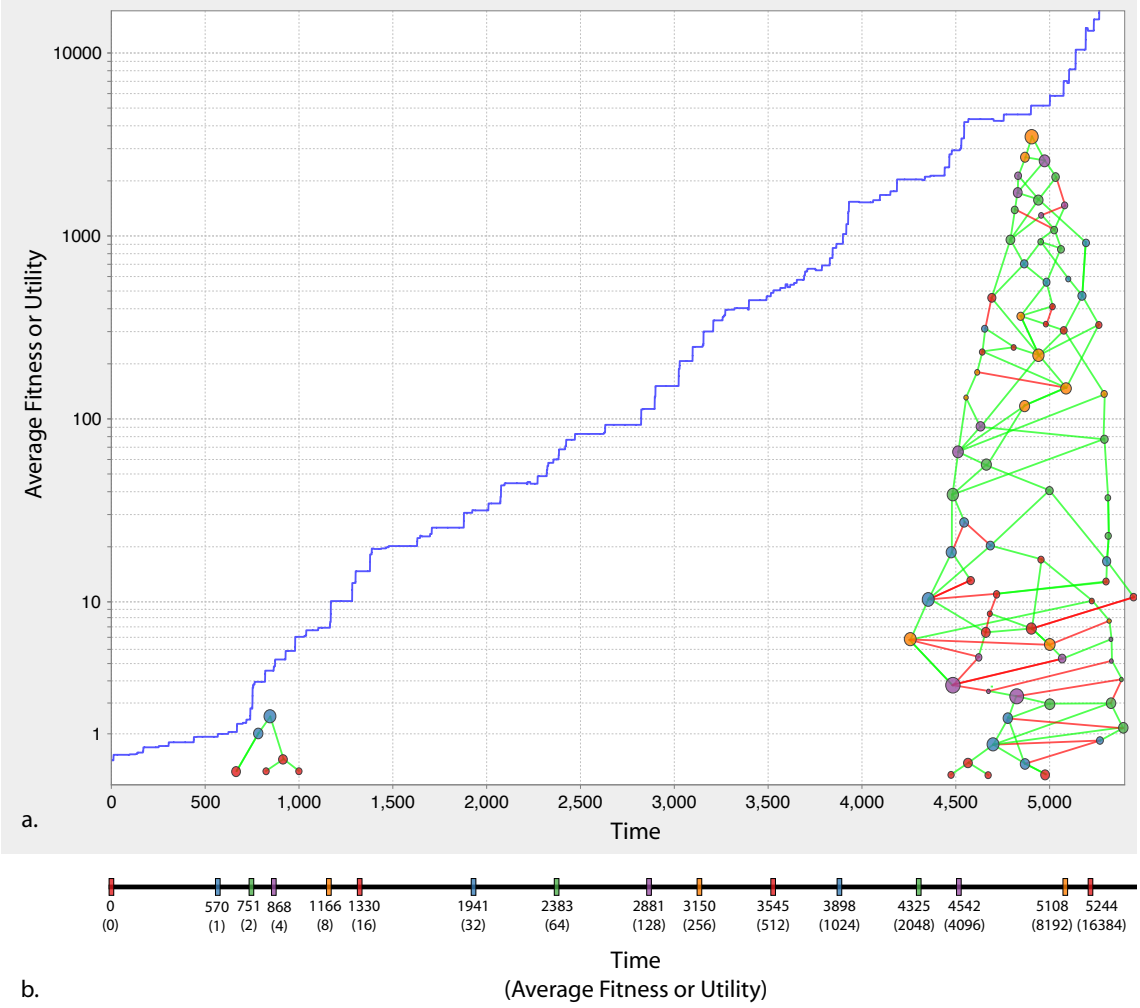


Figure 5.3: Plot of the exponential growth in fitness along with an illustration of the growth in complexity

Figure 5.4 shows the relationship between creative destruction and the average total fitness of the agents in the simulation. The red spikes show time periods where creative destruction occurred. For this simulation run, there were three time periods where a single

new innovation caused three other goods to go 'extinct'. These periods of creative destruction are also associated with a large increase in average total fitness. This simple model was able to demonstrate the creative destruction process by which a new innovation overtakes an existing good(s).

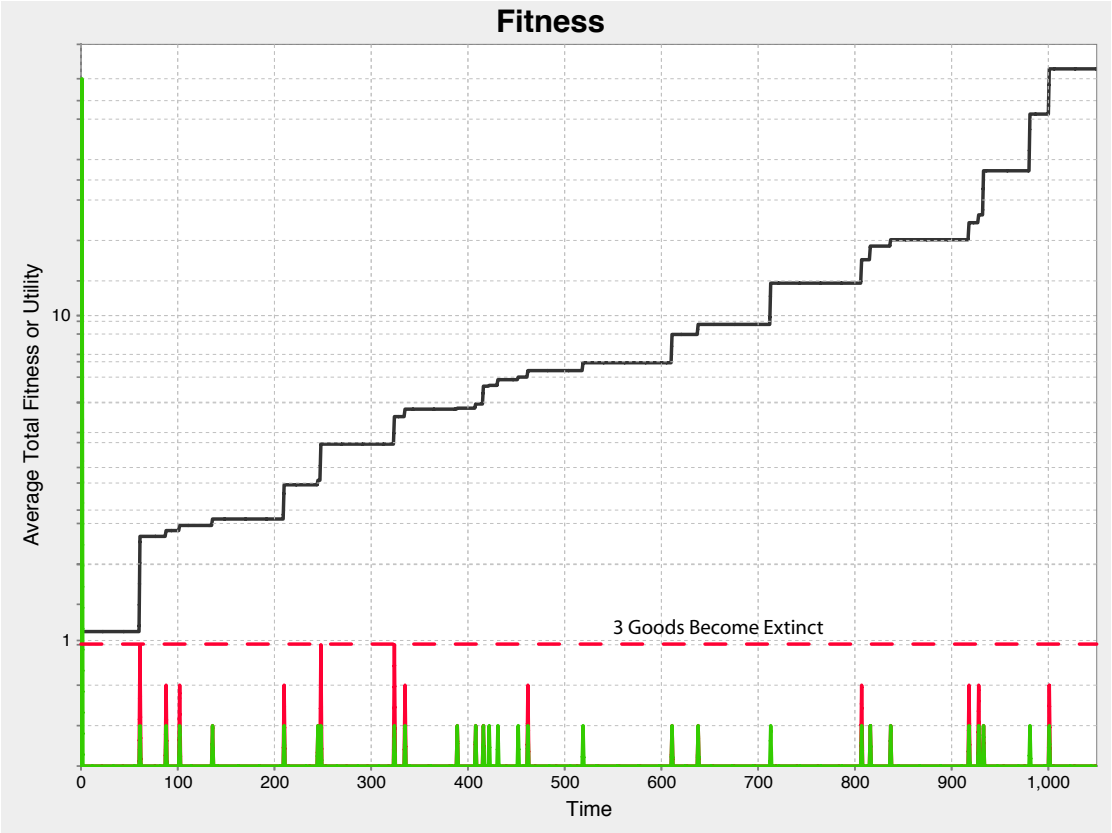


Figure 5.4: Plot showing creative destruction. Red spikes are periods of good extinction and green spikes are when a new good is introduced.

5.2 Relating the Model to Word Usage

This agent-based model is general enough to apply to many types of innovation processes. For example, with regard to word usage, the goods can be viewed as phrases and the components that make up phrases are words. Under this interpretation, a new phrase (previously called a good) is created when two existing phrases are joined together using a newly introduced word. While this may not be realistic linguistically, it allows for a simple mechanism to build from the bottom-up a hierarchical structure of words. This is the same mechanism as described above, but with two variations. The first model variant is the same as above where the initial goods are assigned a fitness from a uniform distribution, while in the second model variant, they are assigned a fitness using a power law:

$$fitness_1(g_i) \in U[0,1] \tag{5.1}$$

$$fitness_2(g_i) = \frac{1}{i} \tag{5.2}$$

Where g_i is the i^{th} good and i is equal to 1 to the number of initial goods (N). However, during the initial distribution of goods to the agents, the probability of getting any particular good is uniform and independent of the good's fitness for both model variants. The model proceeds as discussed above and the agents build new phrases by recombining existing phrases.

In the model results discussed here, there are 25 initial goods (N), while this choice is somewhat arbitrary, Figure 5.5 shows a divergence from Zipf's Law until approximately rank 25 in the empirical word frequency data. This divergence could be caused by those words having special structural importance in the English language. Regardless, the choice in number of initial goods is not critical for the results presented here. A similar set of results are presented in Appendix D where there were 7 initial goods. Also, for all results, there were 1,000 agents (A) each with 1,000 holdings (G), which under this model interpretation are phrases.

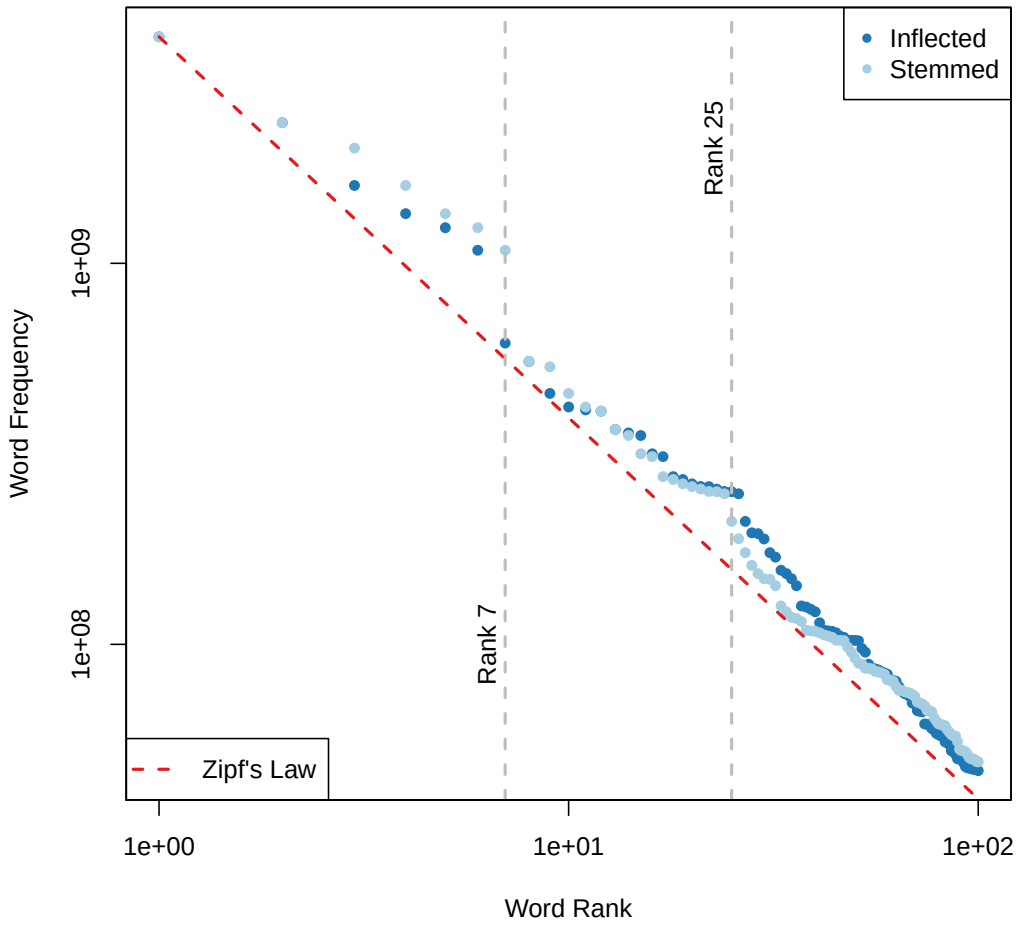
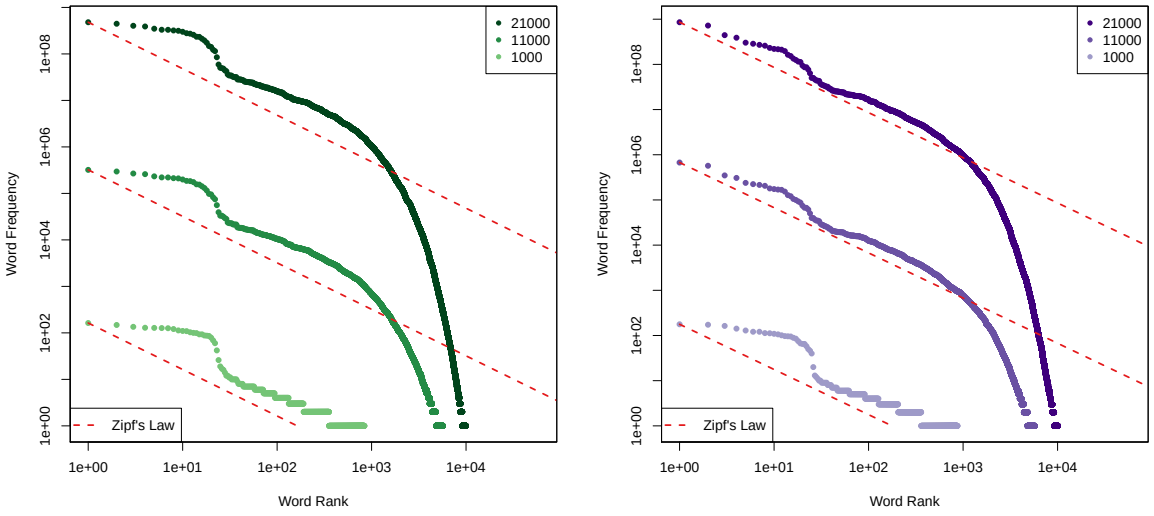


Figure 5.5: Zipf plot showing the first 100 ranks in the empirical word frequency data. There are some deviations from Zipf's Law in the most frequent words.

Figure 5.6 shows the Zipf plot of the active phrases that are currently in the agents' holdings. To be active, a phrase only needs to be held by a single agent. A phrase is a hierarchical structure made of other sub-phrases. For each active phrase, the total uses of words are counted and the sum from all active phrases are displayed in Figure 5.6. If a phrase is held by multiple agents, then its words are only counted once. Figure 5.6 shows how the word usage distribution changes over time by displaying three temporal snapshots of the distribution.



(a) Uniform Initialization

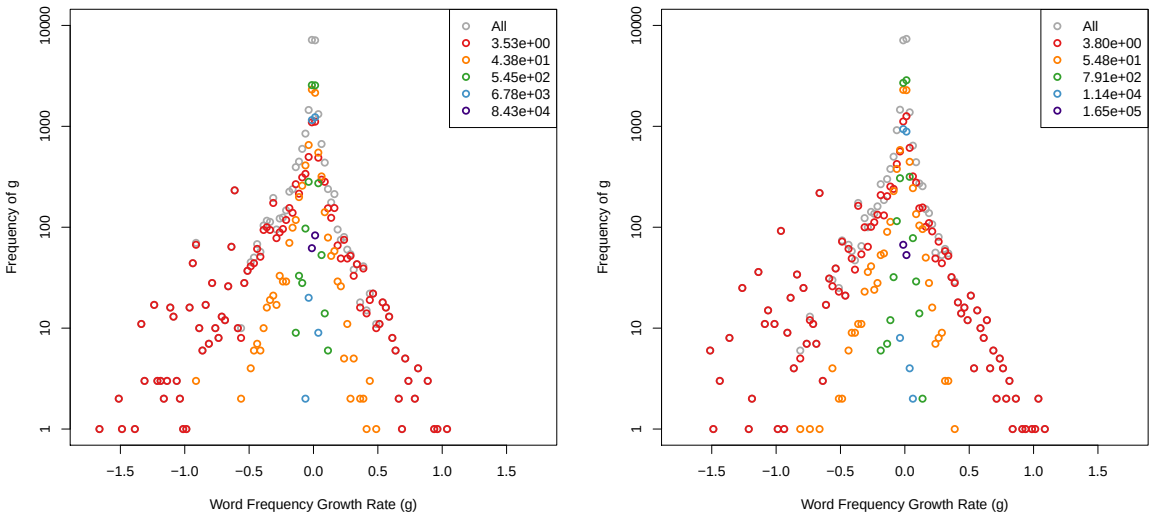
(b) Power Law Initialization

Figure 5.6: Zipf plots for the agent-based model results. For each model variant, there were 25 initial goods that were either assigned fitness values using a uniform distribution or a power law relationship. Each figure shows three snapshots during execution of the model. The legend shows the number of simulation ticks there had been when the data series was collected. Also, since there were on average 1 new phrase per tick, it also gives an estimate of the number of new phrases created.

From Figure 5.6, both variants of the model approximately follow Zipf's Law. Moreover, due to the limited capacity of the agents' holdings, there is a regime change in the agents'

word usage. By using a power law to add some structure to the initial words, the second variant does a better job of following Zipf's Law, especially in the most frequent words (see Figure 5.6b).

The growth rate for each word was calculated every 2,000 ticks starting from tick 11,000 to tick 21,000. Figure 5.7 shows the 'nested' Laplace growth rates from the agent-based model results. This same pattern of 'nested' Laplace growth rates was found in the empirical data in the previous chapter. The growth rate distributions for the agent-based model are noisier than the empirical data, but the agent-based model data had far fewer samples[2]. There appears to be no major difference in the growth rates caused by the different initialization schemes (uniform versus power law).



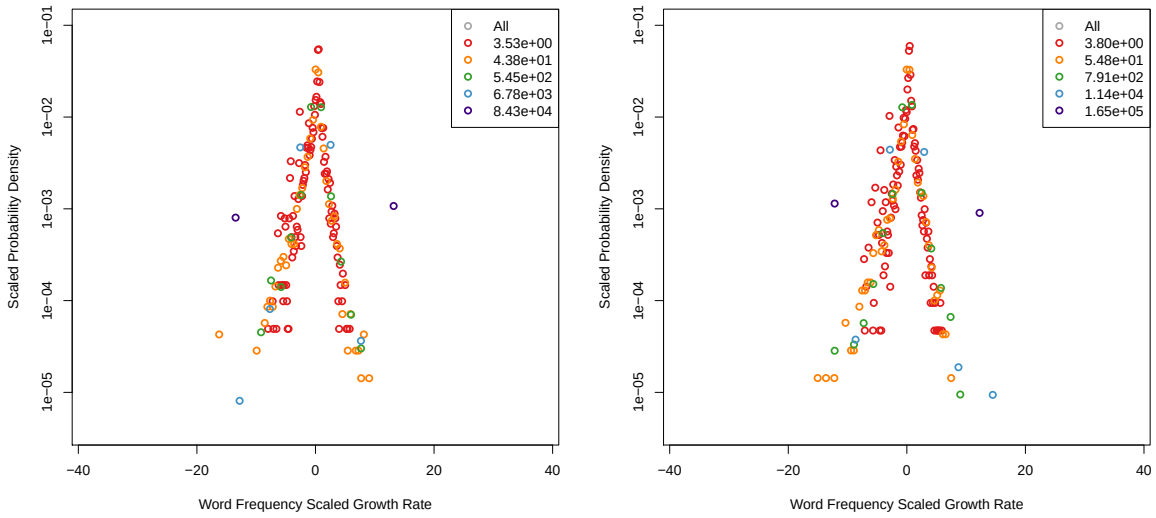
(a) Uniform Initialization

(b) Power Law Initialization

Figure 5.7: Growth rates of word usage for the agent-based model results. The growth rates were calculated for the words in the active phrases every 2,000 ticks starting at tick 11,000 until tick 21,000. The growth rates were logarithmically binned based on the word's frequency at tick 11,000.

[2]The simulations were executed on a machine with 16GB of memory which was the limiting factor on collecting more data.

The growth rates were also rescaled using the same method discussed in the previous chapter (see Section 4.3.1 for details). The rescaled growth rates collapse onto a single curve except for the most frequent words (see Figure 5.8). Since the bin that contains the most frequent words only contain a few words, it is most sensitive to noise in the data, so the failure for these values to rescale is not surprising.



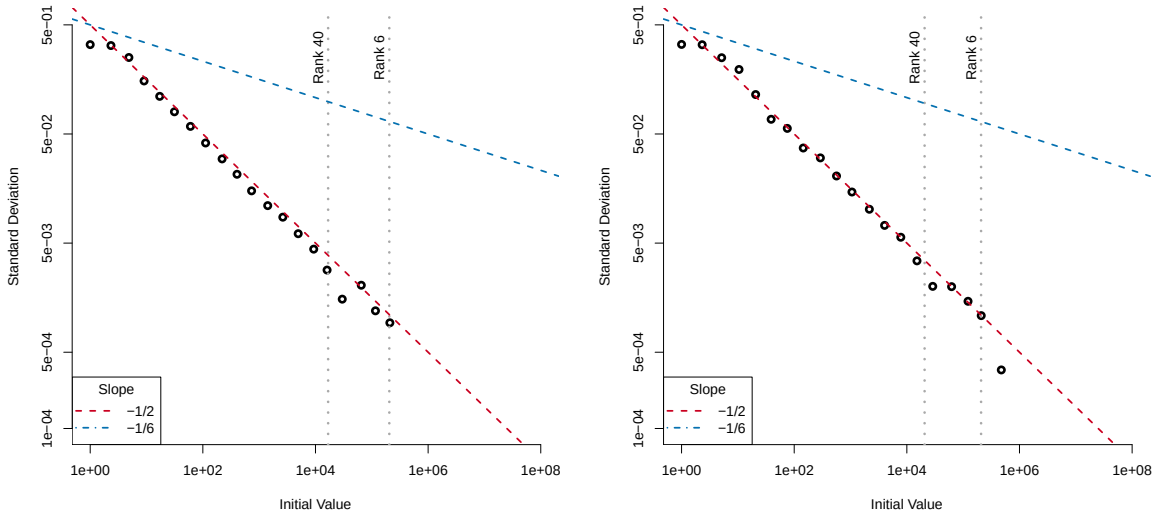
(a) Uniform Initialization

(b) Power Law Initialization

Figure 5.8: Scaled growth rates for the agent-based model results. The growth rates were calculated just like before and then rescaled using the rescaling method discussed in Section 4.3.1 and Stanley et al. [106].

The power law relationship between initial value and standard deviation of the growth rates was also present. The growth rates were calculated the same as before. The initial value was the word frequency at tick 11,000. Both variants of the model have a power law relationship just as in the empirical word frequency data.

Overall this simple agent-based model was able to reproduce the patterns found in the word frequency data. The model's word usage approximated Zipf's Law and included a



(a) Uniform Initialization

(b) Power Law Initialization

Figure 5.9: Power law relationship between initial value and standard deviation for the agent-based model results. The growth rates were calculated every 2,000 ticks from tick 11,000 to tick 21,000. Therefore, the initial value was the word frequency at tick 11,000. Both variants of the model display the power law relationship.

regime change. The model variant with the power law initialization followed Zipf's Law a bit better for the most frequent words. The growth rates of the model data displayed the 'nested' Laplace distribution. These growth rates rescaled fairly well. There was a clear power law relationship between initial values and the growth rates' standard deviation, although the slope of the relationship was steeper than the word frequency data. The slope here is $-\frac{1}{2}$ which means that the relationship is a product of the central limit theorem.

5.3 Discussion

This agent-based model provides a general model of innovation. This model allows for agents' idiosyncratic preferences for some goods over others. Also, the products of the innovation process have a hierarchical structure that grows in complexity over time. Despite

the model's simplicity, it was able to display exponential growth in fitness, creative destruction, and punctuated equilibrium. This chapter also included custom visualizations that helped to better understand the dynamics of the model.

This model is general enough to be applied to many different domains. Word usage was used as an example domain. The model was able to reproduce some of the characteristics of the empirical word usage data, including: Zipf's Law with a regime change, 'nested' Laplace growth rates that can be rescaled, and a power law relationship between initial values and the growth rates' standard deviation. Therefore, this model appears to provide a reasonable abstraction of the process of creating language and it provides a simple mechanism for creating the hierarchical structure of language. However, the slope of the power law relationship between initial values and the growth rates was steeper than the empirical data. Consequently, this provides evidence that the growth rates of the building blocks (words) are more independent of one another in the model than what is found in the empirical word frequency data.

In the next and final chapter, the dissertation will conclude with a brief summary of research findings and some ideas for future work.

Chapter 6: Discussion and Conclusion

This dissertation has provided a tangible way of studying collaborative problem solving by providing quantitative measures of the innovation process under study. The data sets used here have provided valuable insight on a broad spectrum of innovation processes. A summary of the research findings and potential future work are presented. Finally, I conclude with a few final thoughts.

6.1 Summary of Research Findings

This dissertation provides an example of how quantitative approaches can be used to study the process of innovation. Using empirical data analysis and computational models, tangible results were found that can be compared across multiple technological domains. Furthermore, due to the importance of recombination in the innovation process, an emphasis was placed on understanding how building blocks are used. This quantitative approach that focused on building blocks was used to study a broad spectrum of innovation processes (ranging in duration from a week to a century) in diverse domains (software development and word usage in books). In the remainder of this section, I will give a brief overview of research findings from each of the chapters in this dissertation.

Chapters 2 and 3 focused on short-term innovation processes with static fitness landscapes. Chapter 2 provided an example of quantitive data analysis of problem solving, which identified two predictors of innovation: problem modularity and problem size. Chapter 3 showed how a computational model can be used to further understand the problem characteristics discussed in Chapter 2. It also proposed a novel way of validating a computational model using a parameter sweep and symbolic regression. By integrating mathematical and computational models, a better model of innovation was discovered (size divided by

complexity).

Chapters 4 and 5 discussed long-term innovation processes that have dynamic fitness landscapes. Chapter 4 provided evidence that the two power law regimes of word usage is due to a shift from versatile structural words to specialized content words. This chapter also tested two mathematical formulations proposed to express the regime change in word usage. The empirical data provided support for the mathematical formulation that included two power law exponents over the formulation that included a power law with an exponential cutoff. Chapter 4 also showed how the growth rates of words were similar to growth rates found in empirical firm data. Chapter 5 introduced a simple agent-based model that could reproduce patterns seen in innovation processes, including: exponential growth in fitness, creative destruction, and punctuated equilibrium. This agent-based model is general enough to be applied to a variety of domains. Results from the agent-based model were used to study the mechanism behind the empirical findings from the word usage data discussed in Chapter 4. It is proposed that an important part of this mechanism is due to the hierarchical structure of language.

6.2 Future Work

There are many directions that future work can go. Some of the most promising include further development of the model validation technique discussed in Chapter 3. This method of validation attempts to validate a model by creating a good overall fit of the phenomenon under study rather than just matching the particular set of data that is currently available. Using this method, the modeler can be more certain of the model's validity.

There are many possible additions to the agent-based model discussed in Chapter 5. These include adding different roles for agents and placing agents within a social network. Also, the fitness of a good could be made more realistic by making it multidimensional instead of being a single value. This would allow for the modeling of tradeoffs of various performance metrics.

Overall, this dissertation shows how empirical data can be utilized to analyze and model

the process of interest. Creating a balance of research based on empirical findings that is not overly specific to the idiosyncratic properties of the available data is a challenge, but a challenge worth pursuing. Another important future work project is to use other available data sets to see how well the results generalize.

Another innovation process that has much detailed data available is software projects within the open source community. Open source software uses a community-based model of software development where most of the software developers provide their services and software products for free. Therefore, monetary influences can largely be ignored. Any user can then freely download the open source software, and furthermore, they can download the source code for the software if they would like to make modifications. Therefore, every individual can be an innovator and user of the innovations, which is also true for the models described in this dissertation.

The SourceForge Research Data Archive (SRDA), maintained by Greg Madey and fellow researchers at the University of Notre Dame, contains very detailed data about SourceForge.net [189] [190]. SourceForge.net is the largest open source development website in the world and hosts more than 180,000 open source projects and has over 1.9 million registered users [191]. The SRDA contains over 95 monthly snapshots of SourceForge data since 2003. The SRDA contains information about which software developers work on which projects. The SRDA also contains detailed information about when developers are assigned a task and when they commit source code to the project [189].

There have been several data analysis and modeling projects that used data from the SRDA that have a similar character as work presented in this dissertation. For example, various aspects of the innovation process in the open source community have been studied, including: scale-free network structures [192], preferential attachment [193], power laws [85], specialization of contributers' tasks [194] [195]. However, little has been done that focuses on how building blocks come together in this type of environment. For example, Dalle and David [196] studied which modules developers choose to contribute, but did not focus on how the modules were aggregated in the first place. Nevertheless, modularity has been an

important consideration for the design of open source projects. For example, modularity has been central throughout the development of one of the most popular open source projects, Linux. The modularity in Linux reduces the communication between modules and helps portability by keeping specifics of hardware inside a module [197] [198]. This allows code to be written in parallel by multiple programmers in multiple places [199]. The hierarchy of software developers working on Linux developed organically. Linus Torvalds took the lead for the entire project when Linux was first created, but as new modules were incorporated others became leaders of those modules [199]. For example, when Alan Cox contributed a networking module to Linux, he became the lead of all networking issues and Torvalds redirected any networking issues to him [199]. In future work, some of the same quantitative approaches used for the programming contest data may provide further insight in the role of modularity and the use of building blocks within the rich history of open source software.

Research on the open source community may also provide insights into whether innovation is driven by customer demand or innovator supply. Radtke and contributors [200] [201] used modeling to explore what made open source software successful. Their agent-based model found that the driving force in software development was the developer, not the consumer [200] [201]. This falls in line with others accounts of open source development being driven by the developer [202] [203], but against Schmookler's idea that innovation is driven by market demand [34] [42]. Since open source software is being provided at no charge, product demand plays a smaller role than supply. Therefore, future work could use open source projects as prototypical examples of innovation processes that are driven by innovator supply and determine if they evolve differently than innovation processes where the customer has more say in the trajectory of future improvements.

6.3 Conclusion

Simon is frequently quoted on his thoughts about highly skewed distributions:

> "No one supposes that there is any connection between horse-kicks suffered

> by soldiers in the German army and blood cells on a microscopic slide other than that the same urn scheme provides a satisfactory abstract model for both phenomena." [pg 425] [62]

In this dissertation, I have focused on building blocks and how building blocks come together to create higher level modules. These modules can continue to aggregate to form a hierarchical structure. I believe this hierarchical structure has an important role to play in the "urn scheme" that Simon is referring to.

Appendix A: Additional Programming Contest Analysis

A.1 Additional Regression Analysis Results

To ensure that the different scoring criteria did not influence the results of the regression analysis, I include here the regression results for the first thirteen programming contests, where only the result quality and execution time were used in the scoring criteria. These results are displayed in Tables A.1 and A.2 and they present similar results as those discussed in Chapter 2.

Table A.1: Regression results from the first 13 MATLAB contests with innovations per participant as the dependent variable.

	Dependent Variable: Innovations per Participant		
Independent Variables:			
Average Number Functions			
Coefficient	**0.095**		
p-value	**0.007**		
Average Node Count			
Coefficient		**1.8E-04**	
p-value		**0.022**	
Average Maximum Complexity			
Coefficient			0.007
p-value			0.462
Intercept	0.689	0.736	1.022
R-squared	0.494	0.393	0.050
Adjusted R-squared	0.448	0.338	-0.036
F-statistic	10.720	7.131	0.582

Note: Values in **bold** are significant at the 5% level.

Table A.2: Regression results from the first 13 MATLAB contests with percentage of participants that were innovators as the dependent variable.

	Dependent Variable:		
Independent Variables:	Percent Innovators		
Average Number Functions			
Coefficient	**0.013**		
p-value	**0.008**		
Average Node Count			
Coefficient		**2.4E-05**	
p-value		**0.027**	
Average Maximum Complexity			
Coefficient			7.7E-04
p-value			0.554
Intercept	0.178	0.187	0.232
R-squared	0.488	0.373	0.033
Adjusted R-squared	0.441	0.316	-0.055
F-statistic	10.470	6.537	0.373

Note: Values in **bold** are significant at the 5% level.

A.2 Example Parse Trees

To give a better idea of what a MATLAB parse tree is, this section provides a few examples of them and how changes to the code effect similarity values (see Equation 2.4). Table A.3 shows the code for a short function and its corresponding parse tree. Table A.4, shows how a small code change leads to a small change in the parse tree, thereby producing a high similarity value between the two functions (.97). However, Table A.5, shows how another modification to one line of code causes a greater reduction in the similarity value (.77) because this modification caused a shift of an entire sub-tree.

Table A.3: Example parse tree generated from a MATLAB function. The names of variable and function names are not used in the comparison of parse trees, but are shown in the figure.

Code	Parse Tree

```
function [z] = someFunction(x, y)
for i = 1:length(x)
        for j = 1:length(y)
            z(i,j) = x(i) + y(j);
        end
    end
end
```

→

```
FUNCTION:
  ETC:
    ID: (z)
    ETC:
      ID: (someFunction)
      ID: (x)
      ID: (y)
  FOR:
    ETC:
      ID: (i)
      COLON:
        INT: 1
        CALL:
          ID: (length)
          ID: (x)
    FOR:
      ETC:
        ID: (j)
        COLON:
          INT: 1
          CALL:
            ID: (length)
            ID: (y)
      EXPR:
        EQUALS:
          SUBSCR:
            ID: (z)
            ID: (i)
            ID: (j)
          PLUS:
            SUBSCR:
              ID: (x)
              ID: (i)
            SUBSCR:
              ID: (y)
              ID: (j)
```

Table A.4: The parse tree for the same code example as before with a small change in line 4. This small code change resulted in a small change in the parse tree. Using the similarity function (see Equation 2.4), the similarity between this function and the function in Table A.3 is $\frac{35}{36}$, which is approximately .97.

Code	Parse Tree

```
1. function [z] = someFunction(x, y)
2.     for i = 1:length(x)
3.         for j = 1:length(y)
4.             z(i,j) = x(i) * y(j);
5.         end
6.     end
7. end
```

→

```
1  FUNCTION:
2    ETC:
3      ID: (z)
4      ETC:
5        ID: (someFunction)
6        ID: (x)
7        ID: (y)
8    FOR:
9      ETC:
10       ID: (i)
11       COLON:
12         INT:  1
13         CALL:
14           ID: (length)
15           ID: (x)
16     FOR:
17       ETC:
18         ID: (j)
19         COLON:
20           INT:  1
21           CALL:
22             ID: (length)
23             ID: (y)
24       EXPR:
25         EQUALS:
26           SUBSCR:
27             ID: (z)
28             ID: (i)
29             ID: (j)
30           MUL:
31             SUBSCR:
32               ID: (x)
33               ID: (i)
34             SUBSCR:
35               ID: (y)
36               ID: (j)
```

Table A.5: The parse tree for the same code example as before with another change in line 4. This time the change had a larger effect on the parse tree and thereby, lowering the similarity value. Everything is the same in this parse tree as the parse code in Table A.4 up until line 30 of the parse tree at which point everything is different. Therefore, the similarity value of this parse tree with either of the previous two examples is $\frac{29}{mean(36,39)}$, which is approximately .77. If the similarity threshold T was greater than .77, then this function would be classified as a new function type.

Code	Parse Tree

```
1. function [z] = someFunction(x, y)
2.     for i = 1:length(x)
3.         for j = 1:length(y)
4.             z(i,j) = (x(i) * y(j)) ^ 2;
5.         end
6.     end
7. end
```

→

```
1  FUNCTION:
2    ETC:
3      ID: (z)
4      ETC:
5        ID: (someFunction)
6        ID: (x)
7        ID: (y)
8    FOR:
9      ETC:
10       ID: (i)
11       COLON:
12         INT: 1
13         CALL:
14           ID: (length)
15           ID: (x)
16     FOR:
17       ETC:
18         ID: (j)
19         COLON:
20           INT: 1
21           CALL:
22             ID: (length)
23             ID: (y)
24       EXPR:
25         EQUALS:
26           SUBSCR:
27             ID: (z)
28             ID: (i)
29             ID: (j)
30           EXP:
31             PARENS:
32               MUL:
33                 SUBSCR:
34                   ID: (x)
35                   ID: (i)
36                 SUBSCR:
37                   ID: (y)
38                   ID: (j)
39               INT: 2
```

A.3 Additional Results using Higher Similarity Threshold

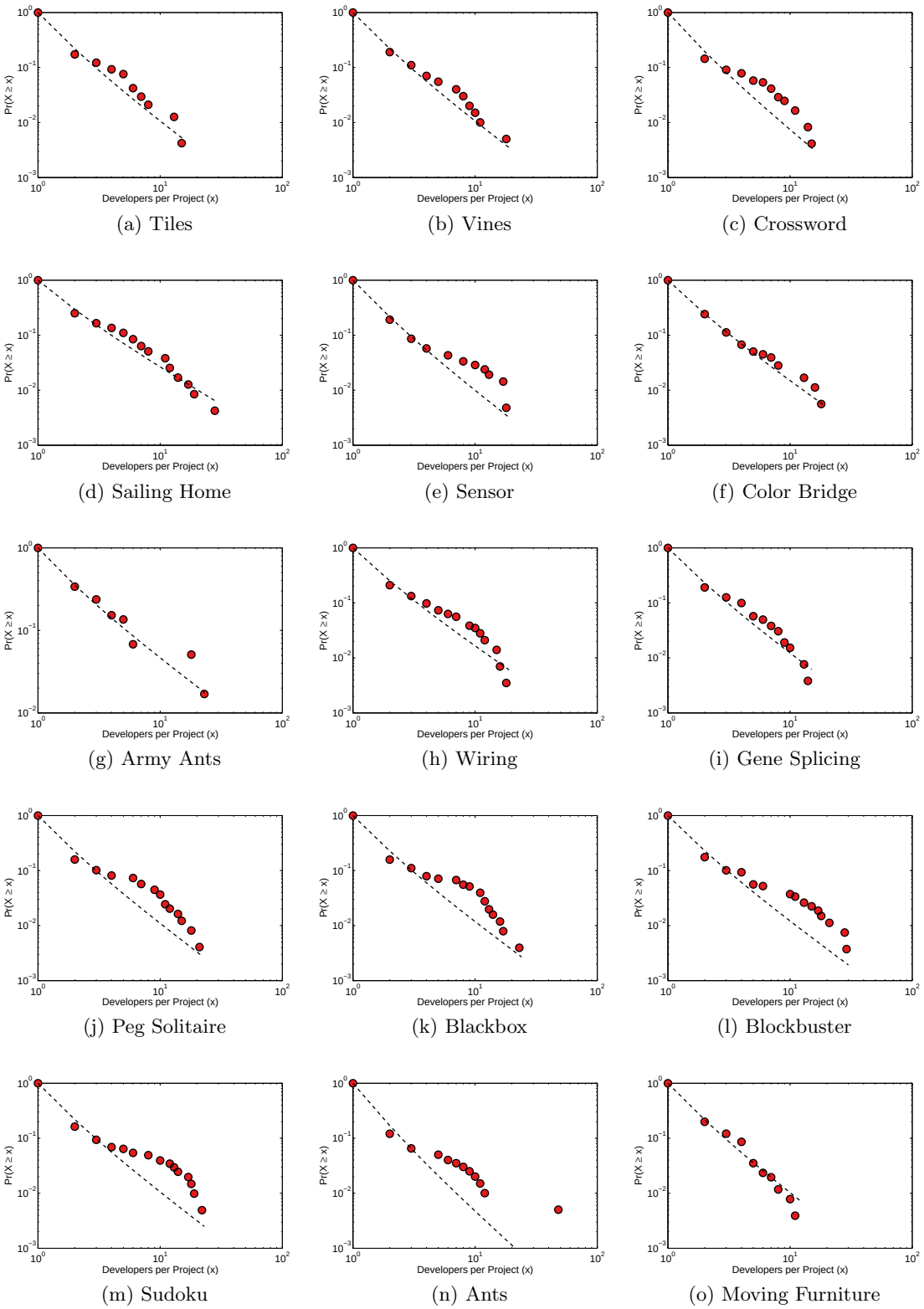
For Chapter 2, the similarity threshold of .70 was used for the analysis. However, another reasonable threshold would have been .85 (see Table 2.2), so included here are the same analyses as Chapter 2, but using a similarity threshold of .85. The results here are very similar to the results presented in Chapter 2.

A.3.1 Additional Distributions of Developers and Projects

Table A.6: Parameters for the developers per project distributions ($T = .85$).

Contest Name	n	$\hat{x}_{min}$	$\hat{\alpha}$	n_{tail}	p
Tiles	237	1	2.68	237	0.00
Vines	199	1	2.68	199	0.07
Crossword	242	1	2.81	242	0.00
Sailing Home	236	1	2.33	236	0.01
Sensor	209	1	2.70	209	**0.16**
Color Bridge	178	1	2.55	178	**0.80**
Army Ants	59	1	2.12	59	**0.73**
Wiring	286	1	2.51	286	0.00
Gene Splicing	262	1	2.62	262	0.00
Peg Solitaire	246	1	2.67	246	0.00
Blackbox	253	1	2.64	253	0.00
Blockbuster	268	1	2.63	268	0.00
Sudoku	203	1	2.68	203	0.00
Ants	199	1	2.99	199	0.00
Moving Furniture	256	1	2.69	256	0.02
Gerrymandering	277	1	2.54	277	0.03
Trucking Freight	114	1	2.06	114	**0.35**
Protein Folding	228	1	2.40	228	0.00
Molecule	153	1	2.71	153	0.02
Mastermind	81	1	2.73	81	**0.13**
Gene Sequencing	53	1	2.96	53	0.02
Mars Surveyor	143	1	2.99	143	0.00
Binpack	217	1	2.99	217	0.03

bold values are statistically significant ($p > .10$)



(a) Tiles (b) Vines (c) Crossword

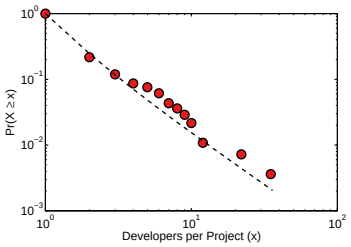
(d) Sailing Home (e) Sensor (f) Color Bridge

(g) Army Ants (h) Wiring (i) Gene Splicing

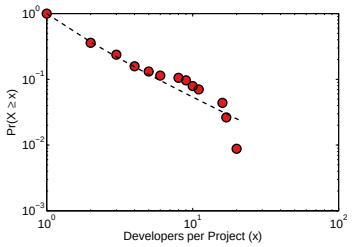
(j) Peg Solitaire (k) Blackbox (l) Blockbuster

(m) Sudoku (n) Ants (o) Moving Furniture

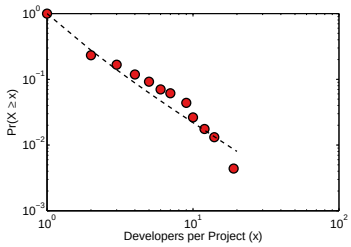
Figure A.1: Developers per project for the 23 programming contests ($T = .85$).



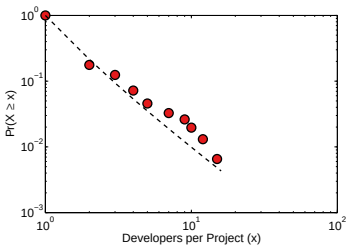
(p) Gerrymandering



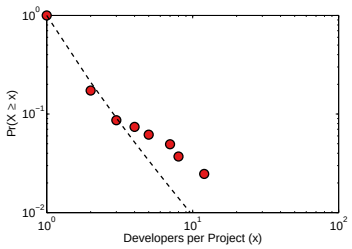
(q) Trucking Freight



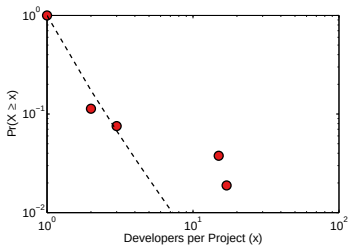
(r) Protein Folding



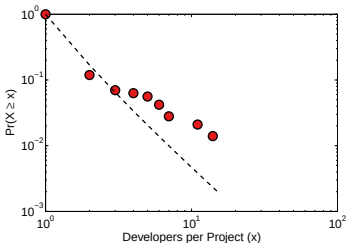
(s) Molecule



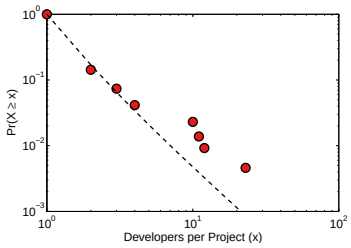
(t) Mastermind



(u) Gene Sequencing

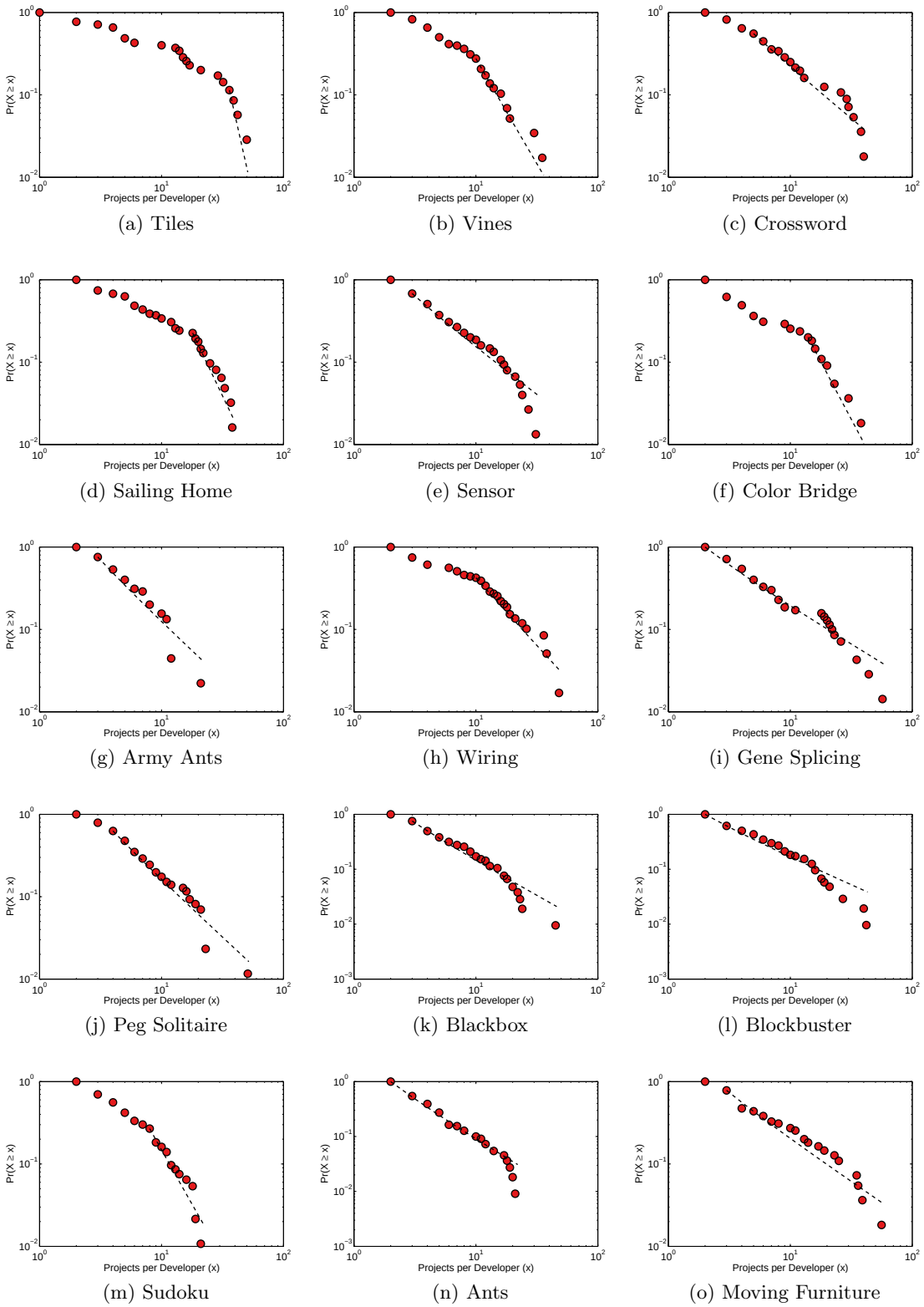


(v) Mars Surveyor



(w) Binpack

Figure A.1: Developers per project for the 23 programming contests ($T = .85$).



(a) Tiles (b) Vines (c) Crossword

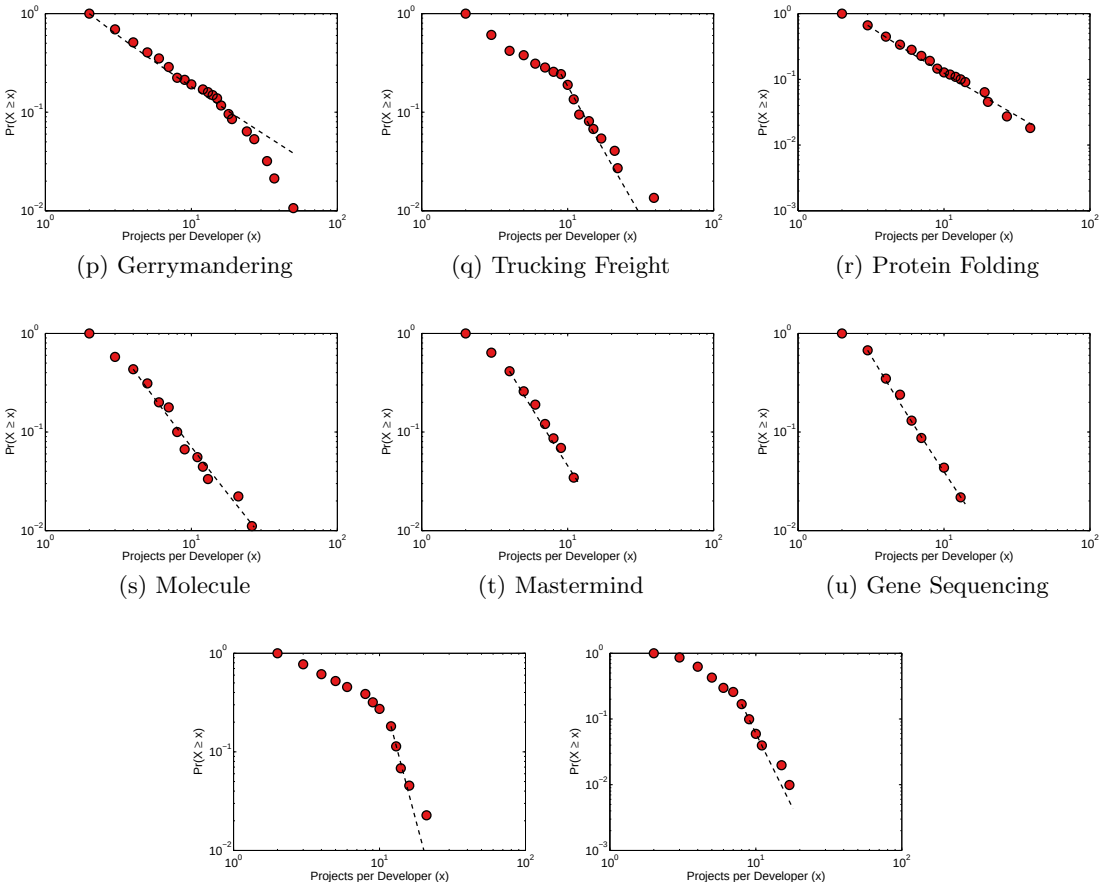
(d) Sailing Home (e) Sensor (f) Color Bridge

(g) Army Ants (h) Wiring (i) Gene Splicing

(j) Peg Solitaire (k) Blackbox (l) Blockbuster

(m) Sudoku (n) Ants (o) Moving Furniture

Figure A.2: Projects per developer for the 23 programming contests ($T = .85$).



(p) Gerrymandering (q) Trucking Freight (r) Protein Folding

(s) Molecule (t) Mastermind (u) Gene Sequencing

(v) Mars Surveyor (w) Binpack

Figure A.2: Projects per developer for the 23 programming contests ($T = .85$).

Table A.7: Parameters for the projects per developer distributions ($T = .85$).

Contest Name	n	$\hat{x}_{min}$	$\hat{\alpha}$	n_{tail}	p
Tiles	35	36	7.52	4	**0.62**
Vines	58	10	3.48	16	**0.82**
Crossword	56	5	2.23	31	**0.53**
Sailing Home	62	18	4.05	14	**0.72**
Sensor	75	3	2.12	51	**0.45**
Color Bridge	55	14	3.74	11	**0.68**
Army Ants	45	3	2.36	34	0.09
Wiring	59	10	2.60	25	**0.61**
Gene Splicing	70	2	1.90	70	**0.17**
Peg Solitaire	86	4	2.36	54	**0.58**
Blackbox	105	3	2.24	79	0.04
Blockbuster	104	2	1.98	104	0.01
Sudoku	93	8	3.53	25	**0.31**
Ants	110	2	2.33	110	**0.16**
Moving Furniture	55	3	2.01	43	0.03
Gerrymandering	94	2	1.94	94	**0.18**
Trucking Freight	74	9	3.55	18	**0.66**
Protein Folding	110	3	2.27	73	**0.45**
Molecule	90	4	2.83	39	0.10
Mastermind	58	4	3.23	24	**0.17**
Gene Sequencing	46	3	3.15	31	**0.47**
Mars Surveyor	44	2	1.86	8	0.01
Binpack	101	8	5.34	17	**0.75**

bold values are statistically significant ($p > .10$)

A.3.2 Evidence of Creative Destruction

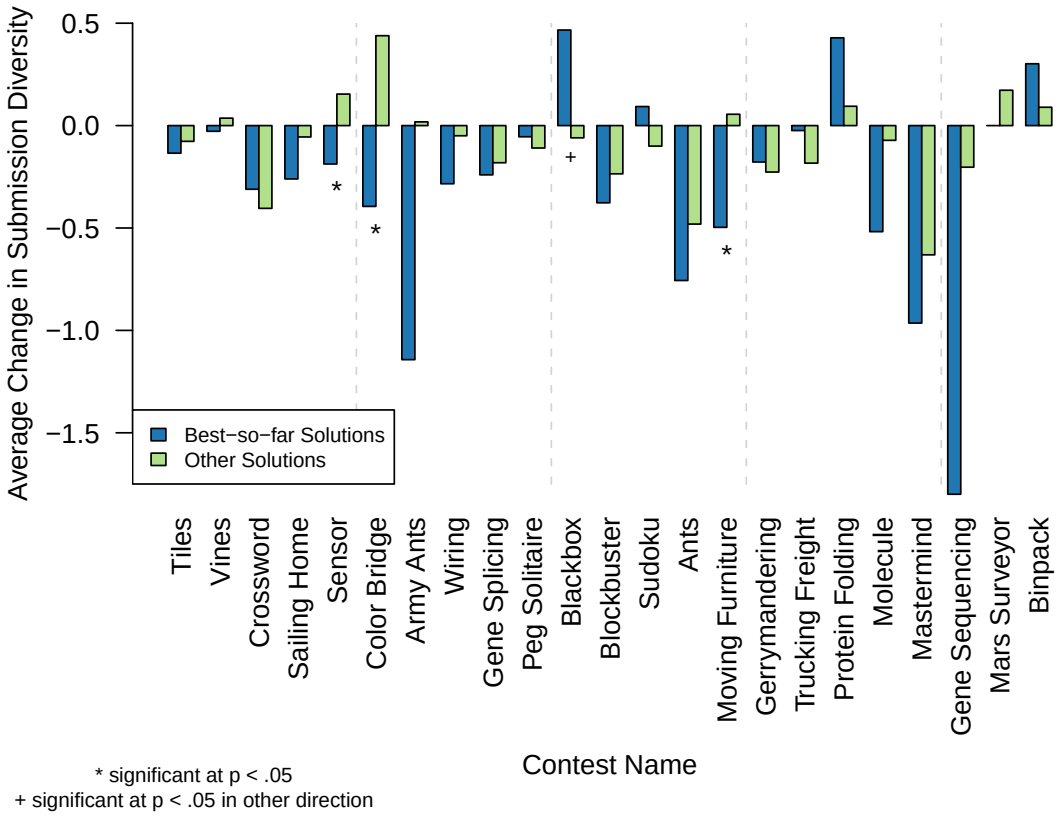


Figure A.3: Creative destruction in the programming contests using the change in submission diversity with the higher similarity threshold ($T = .85$). Fewer contests (14 of 23) have a drop in submission diversity compared to the results with the lower similarity threshold (.70), due to it being easier to start a new approach with the stricter similarity threshold.

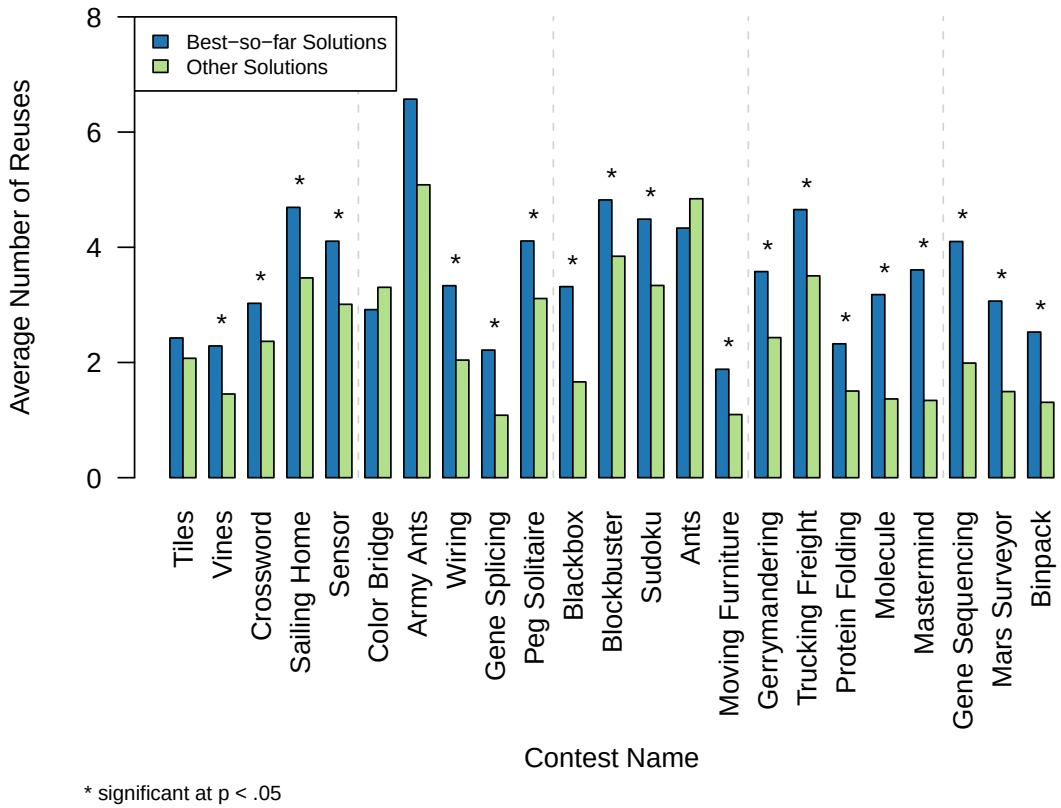
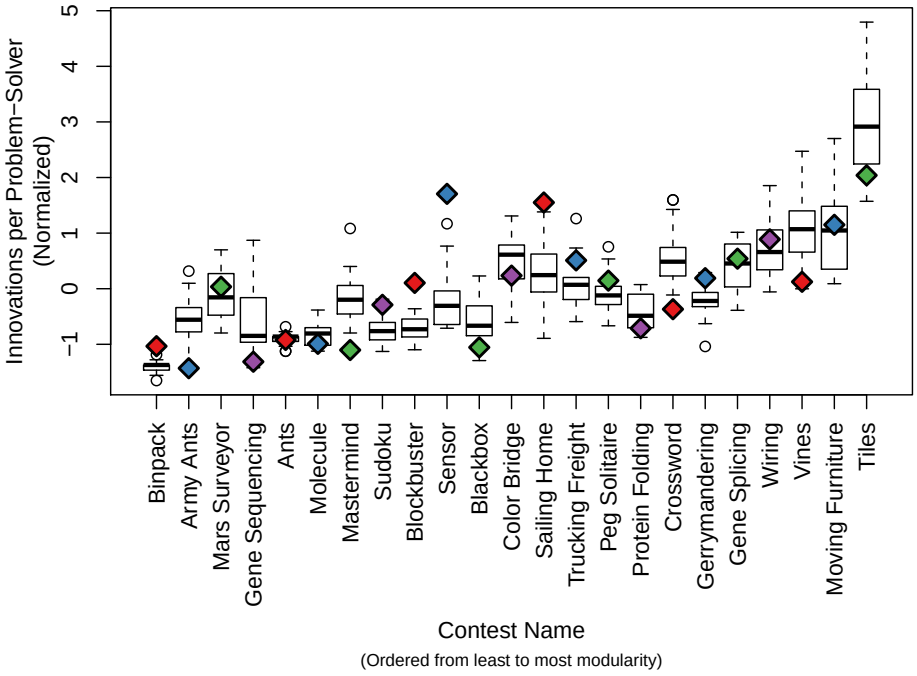


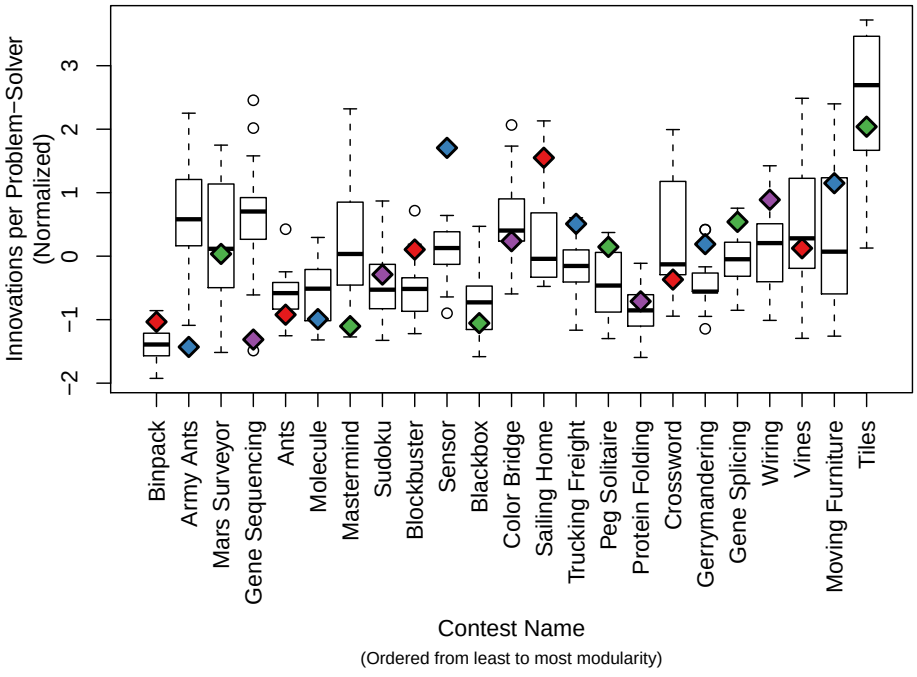
Figure A.4: Submission reuse in the 23 programming contests with the higher similarity threshold ($T = .85$). Most contests (19 of 23) have a statistically significant difference in the amount of reuse between best-so-far submissions and other submissions.

Appendix B: Modifying Selection Pressure in Genetic Algorithm Model

To ensure the genetic algorithm model, discussed in Chapter 3, was not overly sensitive to the selection pressure used, results from three tournament sizes were gathered. Tournament selection allows for the systemic adjustment of the selection pressure. As the tournament size increases, the selection pressure becomes stronger. As seen in these results, the genetic algorithm model results are not dependent on a specific level of selection pressure.

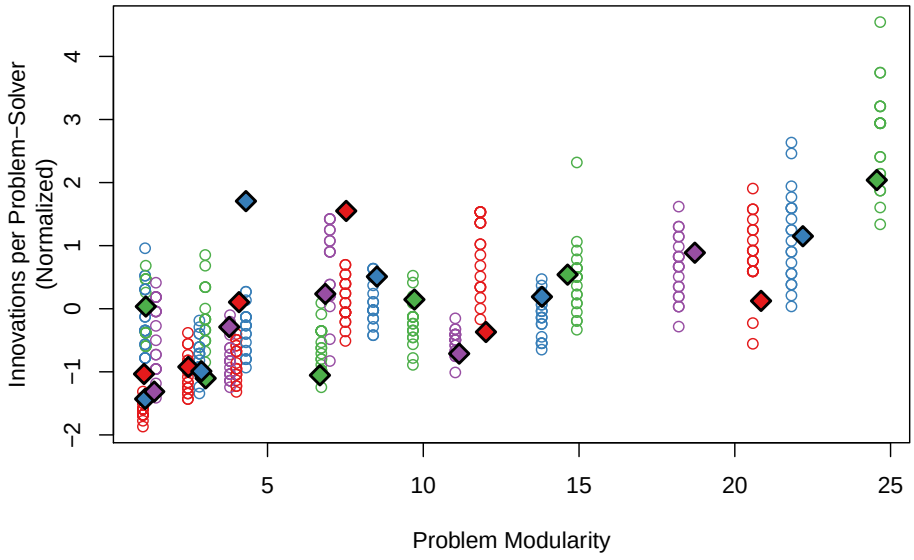


(a) Adjacent Landscape



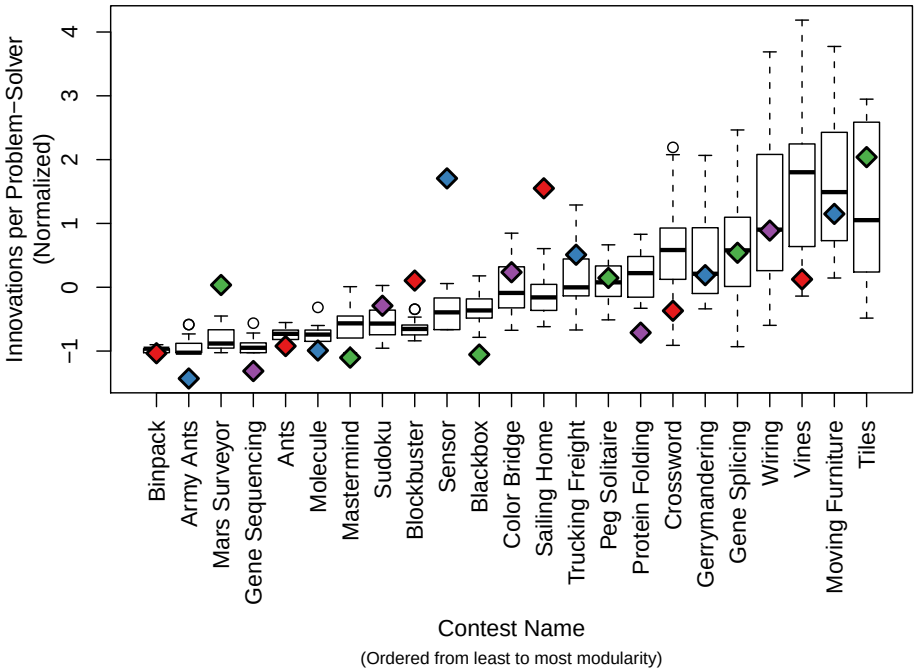
(b) Random Landscape

Figure B.1: Results for the genetic algorithm model for recreating the programming contests (tournament size = 2). These two landscapes are the ones included in Kauffman's original model [49].

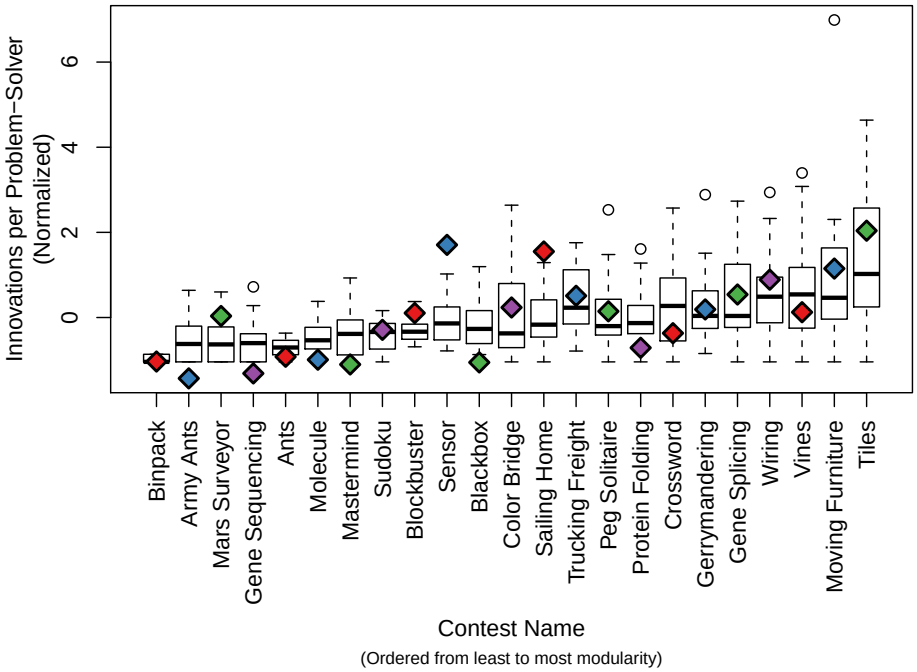


(c) Modular Landscape (Scatterplot)

Figure B.1: Results for the genetic algorithm model for recreating the programming contests. The tournament size for selection was 2, the smallest selection pressure collected.

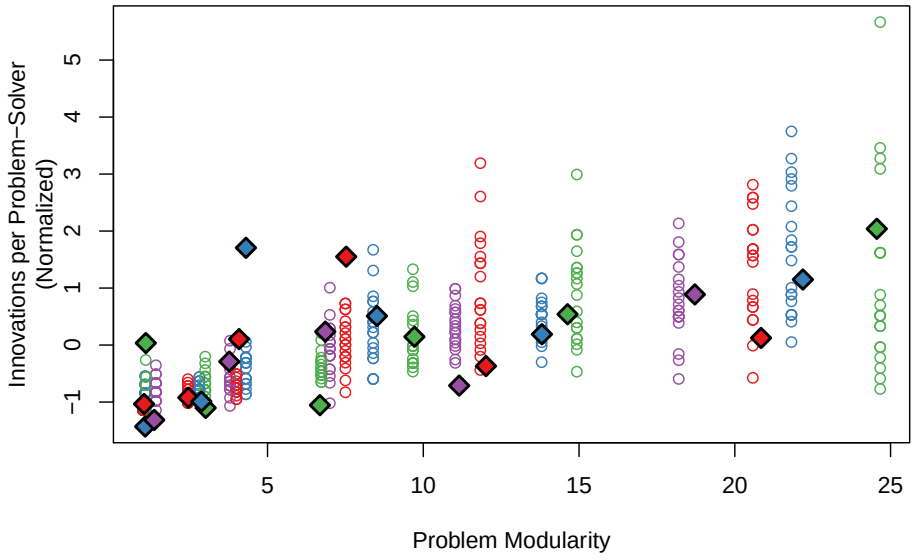


(a) Adjacent Landscape



(b) Random Landscape

Figure B.2: Results for the genetic algorithm model for recreating the programming contests (tournament size = 12). These two landscapes are the ones included in Kauffman's original model [49].



(c) Modular Landscape (Scatterplot)

Figure B.2: Results for the genetic algorithm model for recreating the programming contests. The tournament size for selection was 12, the largest selection pressure collected.

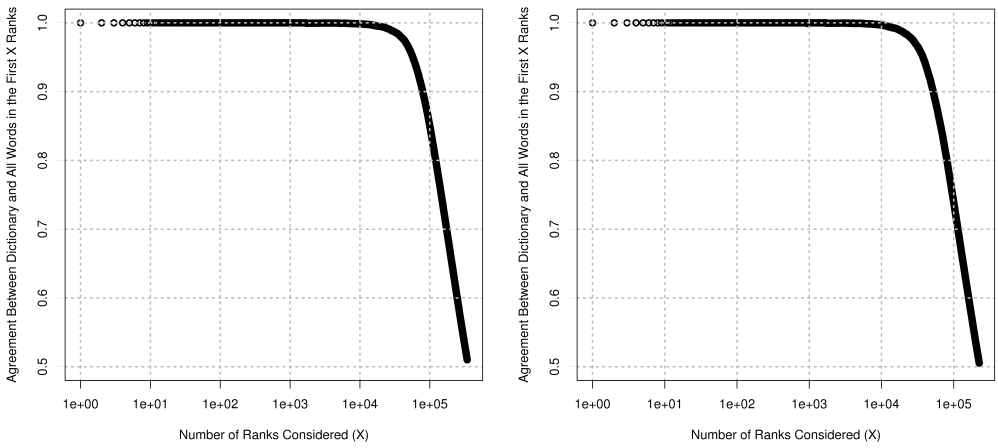
Appendix C: Additional Word Frequency Analysis

C.1 Agreement Between of Words in Top Frequencies

In Chapter 4, various subsets of the word frequency data were displayed separately. To ensure that the regime change occurred with the same set of words in the top ranks (i.e., the first regime), the percentage agreement between all of the words and the subsets of words were calculated. The inflected subsets were compared against all inflected words and the same goes for the stemmed words. Using an agreement threshold of 95%, Table C.1 clearly shows that the regime changed with the same words in the first regime. Therefore, the regime change in the subsets of the data did not simply occur due to the tail of the distribution being moved to the left.

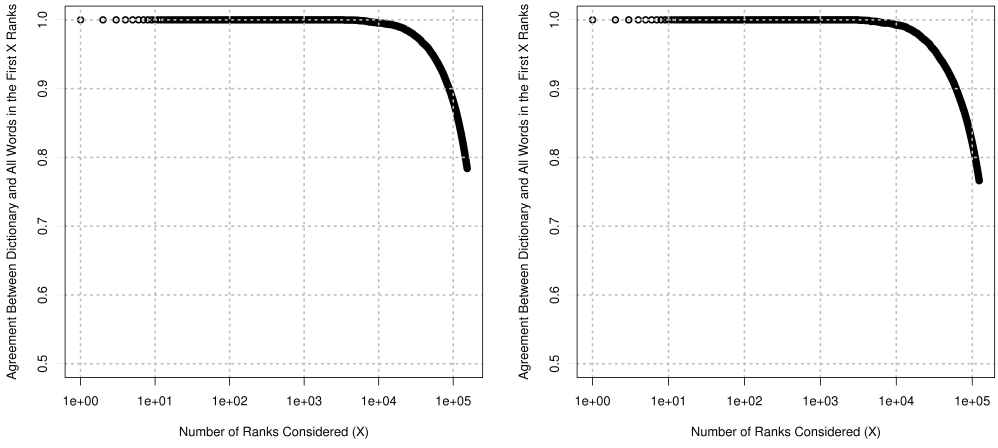
Table C.1: Last rank to achieve the specified percentage agreement.

	Last Rank to Achieve Percent Agreement with All Words	
Subset of Words	95%	99%
Dictionary - Inflected	57,163	27,152
Dictionary - Stemmed	37,591	17,418
All Years - Inflected	52,784	18,605
All Years - Stemmed	34,551	13,108
Dictionary/All Years - Inflected	38,016	14,938
Dictionary/All Years - Stemmed	24,863	9,778



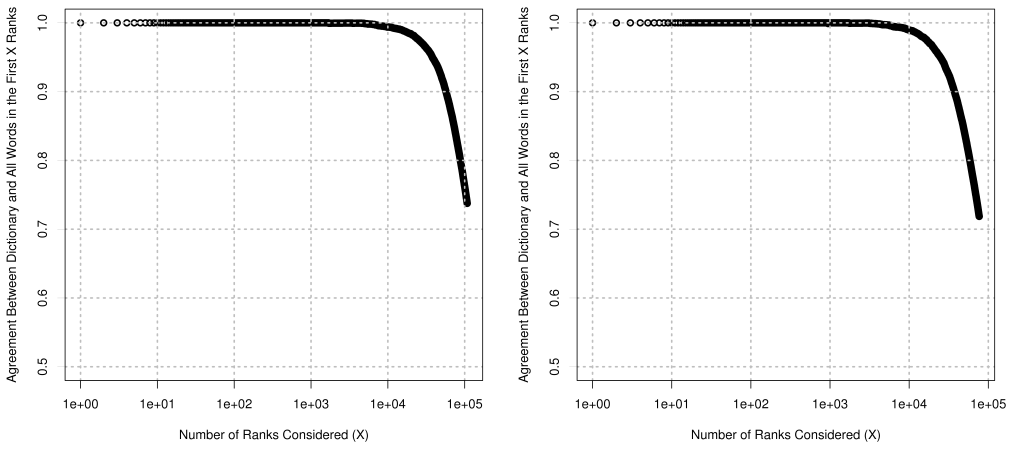
(a) Inflected Words (b) Stemmed Words

Figure C.1: This figure shows the agreement between dictionary and all words in the first ranks. To test if there was differences in the data when just dictionary words were considered, a comparison between all words and the dictionary words for the first X ranks was performed. There was 99% agreement for the first 27,152 ranks for the inflected words and for the first 17,418 ranks for the stemmed words.



(a) Inflected Words (b) Stemmed Words

Figure C.2: This figure shows the agreement between words that appear every year and all words in the first ranks. This analysis tests how much of the second regime is made up of words that do not appear every year. There was 99% agreement for the first 18,605 ranks for the inflected words and for the first 13,108 ranks for the stemmed words.



(a) Inflected Words (b) Stemmed Words

Figure C.3: This figure shows the agreement between dictionary words that appear every year and all words in the first ranks. This analysis tests how much of the second regime is made up of dictionary words that do not appear every year. There was 99% agreement for the first 14,938 ranks for the inflected words and for the first 9,778 ranks for the stemmed words.

C.2 Estimated Parameters Using Maximum Likelihood

To ensure that the parameter estimates were correct, the estimates were also calculated using maximum likelihood estimation. The maximum likelihood estimator used the "L-BFGS-B" method [204]. Maximum likelihood estimation requires a specification of the noise in the data, which was chosen to be Gaussian. The nonlinear least squares estimates presented in Chapter 4 does not require a specification of noise in the data. When calculating the AIC, nonlinear least squares uses the data to calculate the likelihood function, but the noise function is used in the likelihood calculation of the maximum likelihood estimator. Therefore, the AIC values for the maximum likelihood estimator and the nonlinear least squares results are going to be somewhat different, although the AIC values will still be comparable within one method of parameter estimation.

In addition to the difference in AIC values, there is one other difference in the results presented here that is totally superficial, which has to do with the interpretation of the

parameter estimates of the power-power model. For the power-power model, there will always be two parameter estimates that fit the data best. The two sets of parameter estimates are equivalent where the meaning of the r and q values are swapped and μ is now equal to $1 - \mu$. The maximum likelihood estimator found one set of parameter estimates that fit the data best and the nonlinear least squares results presented in Chapter 4 found the other. Therefore, to compare the power-power parameter estimates reported here to the estimates from Chapter 4, swap r and q, and $\mu = 1 - \mu$. Other than these two minor differences (i.e., the AIC values generated and the interpretation of the parameter estimates of the power-power model), the results are very similar. Just as in the results from Chapter 4, the power-power model outperforms the power-exponential model.

Table C.2: Parameter estimates for the two models (power-power and power-exponential) when using maximum likelihood estimation.

	power-power						power-exponential				
	parameters			fit quality		I.C.	parameters		fit quality		I.C.
Data Set	r	q	μ	R	D	AIC	γ	b	R	D	AIC
All Words											
Inflected	2.06	1.49	0.99	.998	.041	66.6	1.61	5.30	.995	.069	75.0
Stemmed	2.07	1.51	0.98	.997	.053	67.3	1.60	4.80	.994	.074	73.3
Dictionary											
Inflected	2.05	1.27	0.998	.989	.092	70.4	1.41	9.38	.982	.129	100.6
Stemmed	1.96	1.22	0.998	.985	.106	71.2	1.37	9.61	.975	.146	101.9

Note:
$-1 \leq R \leq 1$ and larger values are better.
$0 \leq D \leq 1$ and smaller values are better.
For AIC, smaller values are better.

C.3 Top 50 Ranked Words by Part-of-Speech

The performance of the Stanford part-of-speech tagger[1] [179] [180] appears to be quite robust (see Table C.3 and C.4). However, for the nouns, there are some single letter words that appear be mistakes, but are partly due to the way the ngram data was processed. From Table C.3, we see 't', 'p', 'd', 'm', 'c', and 'b' in ranks 2, 15, 25, 39, 41, and 44, respectively. When Google processed the 1-grams, a sequence of characters would be split when an apostrophe is found [70]. Therefore, the contraction "don't" would be broken into three 1-grams: "don", the apostrophe, and "t". This would also hold for other contractions (e.g., won't, can't, I'm, I'd). It is also worth noting that when processing apostrophe 's', the characters were treated as a single 1-gram (e.g., "it's" and "John's" were treated as a single 1-gram) [70].

Other problems come from abbreviations (e.g., p for page, m for meter, t for ton or temperature, M.D., M.S., B.A. and Ph.D. for degrees, A.D. and B.C. for eras, or A.M. or P.M. for time). This could also include abbreviations for people's first or middle names. Hyphenated words could also cause this problem (e.g., p-value, t-test, B-52). It should also be noted that nonsensical words are labeled as a noun (e.g., the random sequence "ilmwecsm" would be labeled as a noun). With a data set of this size, some errors are expected, but overall, the words tagged with part-of-speech appear to be quite reliable.

[1]The Stanford NLP libraries can be found here: http://nlp.stanford.edu/software/

Table C.3: Top 50 ranks for inflected words categorized by part of speech.

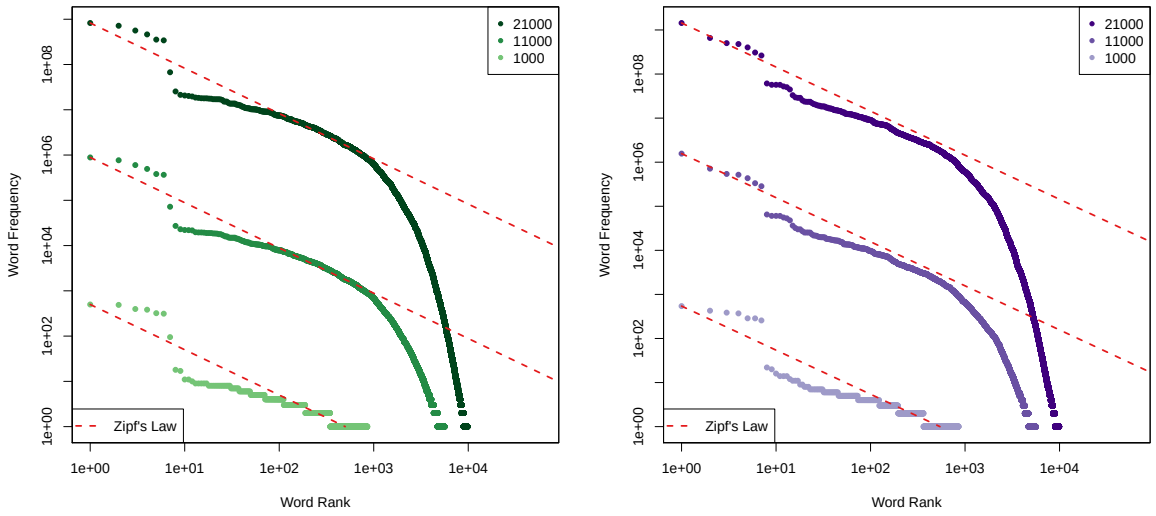
	Part-of-Speech							
Rank (r)	Noun	Conj. /Prep.	Verb	Det.	Adj.	Adv.	Pron.	WH
1	time	of	is	the	more	not	it	which
2	t	and	was	a	other	there	he	when
3	man	to	be	this	such	so	his	who
4	work	in	are	an	new	only	they	what
5	life	that	had	all	most	first	their	where
6	years	for	have	no	many	up	we	how
7	people	as	were	these	great	also	you	whose
8	state	with	been	some	same	then	its	why
9	way	by	has	any	good	very	her	whom
10	part	on	would	those	own	now	him	whatever
11	men	or	will	each	general	well	them	whenever
12	day	at	may	both	old	even	she	wherever
13	world	from	can	another	high	much	my	wherein
14	use	but	should	every	large	little	our	whoever
15	p	if	made	la	small	however	me	whichever
16	case	than	do	del	less	long	us	
17	government	into	could	th	present	here	your	
18	states	out	see	dem	social	down	himself	
19	number	about	said	theses	few	still	itself	
20	place	after	must	nary	different	too	themselves	
21	war	between	being		last	never	myself	
22	power	over	did		second	right	herself	
23	water	like	used		certain	just	ourselves	
24	system	upon	make		important	thus	yourself	
25	d	through	found		possible	back	yours	
26	form	under	given		political	again	ours	
27	year	before	might		human	far	thine	
28	order	because	shall		national	yet	ya	
29	point	without	does		several	often		
30	american	while	know		young	always		
31	fact	during	take		best	later		
32	law	against	come		necessary	once		
33	public	de	came		common	therefore		
34	house	per	called		better	rather		
35	end	since	say		true	early		
36	school	mr	go		white	almost		
37	country	among	united		english	away		
38	hand	though	left		various	further		
39	m	within	thought		least	ever		
40	children	until	give		free	perhaps		
41	c	off	set		economic	south		
42	whole	above	following		full	already		
43	course	although	taken		next	quite		
44	b	whether	get		british	together		
45	york	either	find		special	usually		
46	history	nor	known		due	enough		
47	means	et	think		french	north		
48	others	along	am		short	especially		
49	god	near	having		able	sometimes		
50	group	st	cannot		real	soon		

Table C.4: Top 50 ranks for stemmed words categorized by part of speech.

	Part-of-Speech							
Rank (r)	Noun	Conj. /Prep.	Verb	Det.	Adj.	Adv.	Pron.	WH
1	time	of	be	the	other	not	it	which
2	man	and	have	a	more	there	he	when
3	use	to	do	this	such	so	his	who
4	state	in	make	an	new	only	they	what
5	work	that	will	all	most	first	their	where
6	year	for	would	no	many	up	we	how
7	day	as	may	these	great	also	you	whose
8	t	with	say	some	same	then	its	why
9	life	by	see	any	own	very	her	whom
10	part	on	can	those	good	even	him	whatever
11	case	or	give	each	general	now	them	whenever
12	form	at	take	both	present	well	she	wherever
13	way	from	go	another	old	much	my	wherein
14	place	but	come	every	high	long	our	whoever
15	show	if	should	la	large	little	me	whichever
16	people	than	know	del	small	however	us	
17	point	into	find	th	last	here	your	
18	number	out	could	dem	less	down	himself	
19	child	about	must	nary	social	still	itself	
20	power	after	think		few	too	myself	
21	hand	between	become		different	back	herself	
22	world	over	follow		second	never	yourself	
23	system	like	call		certain	right	yours	
24	order	upon	get		important	just	ours	
25	change	through	write		open	thus	thine	
26	leave	under	seem		possible	again	ya	
27	school	before	begin		human	far		
28	interest	because	look		political	yet		
29	government	de	might		national	often		
30	law	without	shall		several	later		
31	study	mr	hold		common	once		
32	result	while	tell		young	therefore		
33	house	during	feel		best	rather		
34	p	against	set		necessary	early		
35	country	per	bring		better	almost		
36	thing	since	live		true	away		
37	group	among	appear		white	further		
38	water	though	unite		free	close		
39	end	within	turn		english	ever		
40	war	until	provide		various	south		
41	fact	off	include		least	already		
42	line	above	keep		economic	quite		
43	word	toward	mean		full	together		
44	need	although	meet		next	usually		
45	american	whether	want		british	enough		
46	act	either	put		due	north		
47	increase	nor	consider		special	sometime		
48	d	et	require		complete	especially		
49	value	along	stand		clear	soon		
50	woman	near	produce		direct	probably		

Appendix D: Additional Figures for the Agent-Based Model

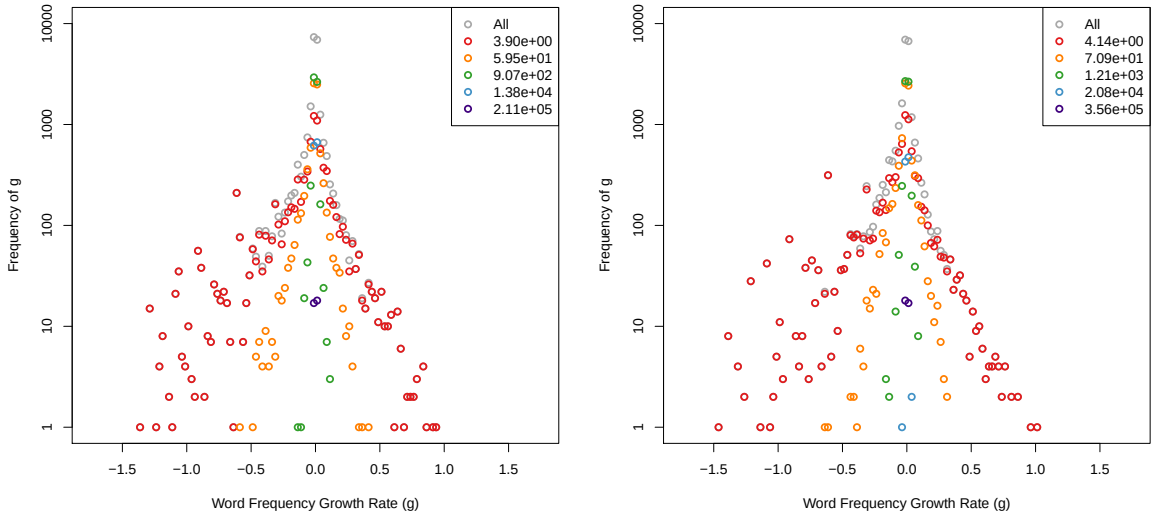
This appendix displays the same figures shown in Chapter 5, but with 7 initial goods instead of 25. The results are very similar to that discussed in Chapter 5, but are included here to show how the number of initial goods influence the model results. The number of initial goods influence the size of the bump in the Zipf plot for the most frequent words. Otherwise, the results presented here are very similar to the ones presented in Chapter 5.



(a) Uniform Initialization

(b) Power Law Initialization

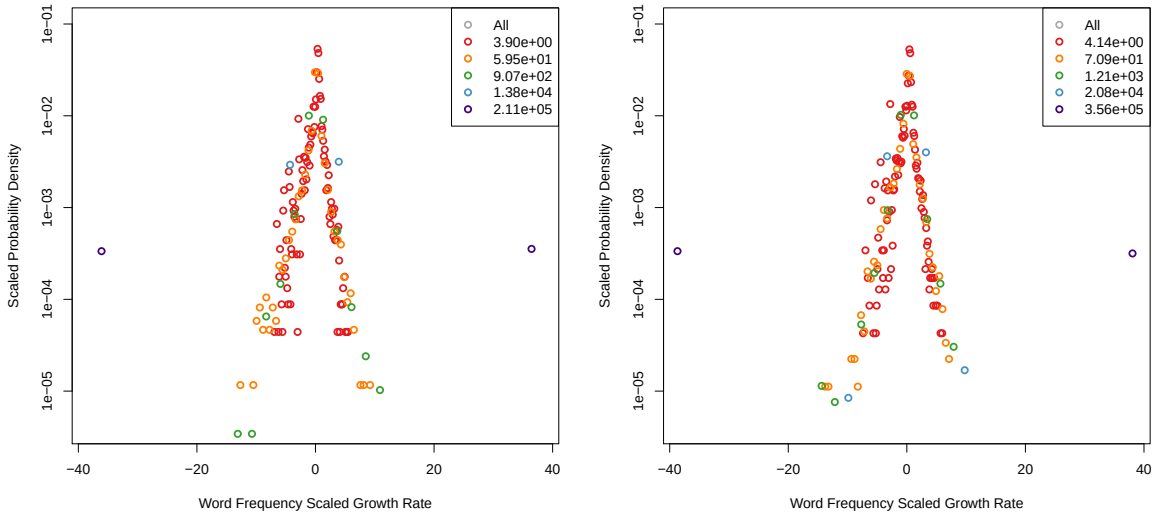
Figure D.1: Zipf plots for the agent-based model results. For each model variant, there were 7 initial goods that were either assigned fitness values using a uniform distribution or a power law relationship. Each figure shows three snapshots during execution of the model. The legend shows the number of simulation ticks there had been when the data series was collected. Also, since there were on average 1 new phrase per tick, it also gives an estimate of the number of new phrases created.



(a) Uniform Initialization

(b) Power Law Initialization

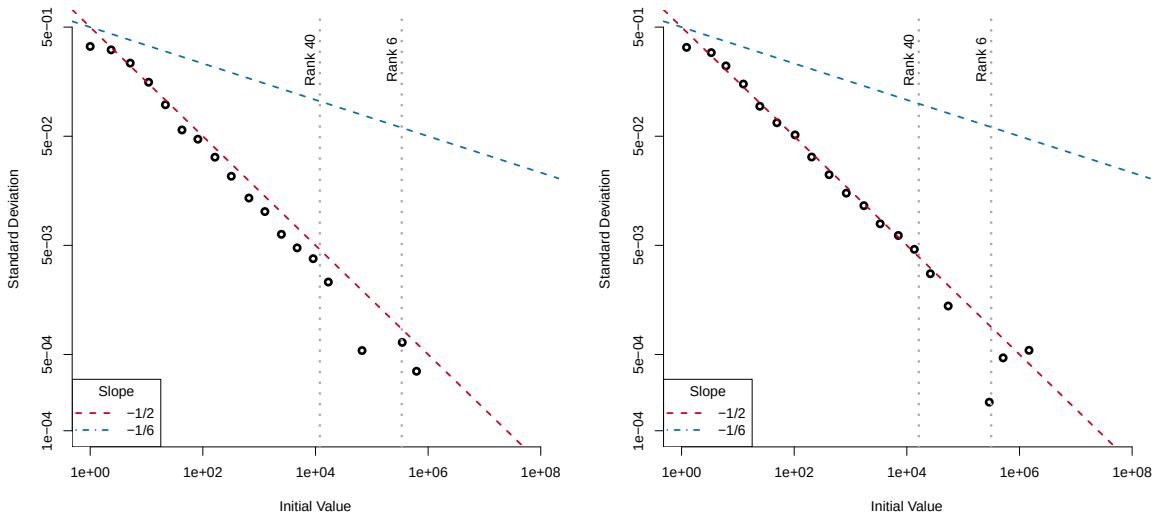
Figure D.2: Growth rates of word usage for the agent-based model results when using 7 initial goods. The growth rates were calculated for the words in the active phrases every 2,000 ticks starting at tick 11,000 until tick 21,000.



(a) Uniform Initialization

(b) Power Law Initialization

Figure D.3: Scaled growth rates for the agent-based model results when using 7 initial goods. The growth rates were calculated just like in Figure D.2 and then rescaled using the rescaling method discussed in Section 4.3.1 and Stanley et al. [106].



(a) Uniform Initialization

(b) Power Law Initialization

Figure D.4: Power law relationship between initial value and standard deviation for the agent-based model results when using 7 initial goods. The growth rates were calculated every 2,000 ticks from tick 11,000 to tick 21,000. Therefore, the initial value was the word frequency at tick 11,000. Both variants of the model display the power law relationship. It is worth noting that some of the bins in the figures do not contain any words, so there are a few missing points (two for the left figure and one for the right).

Appendix E: Power Laws Explained

In this dissertation, there are two formulations of power laws (or power functions) used and it is worth noting why there is a discrepancy. Furthermore, this section is provided to help readers not familiar with power laws to gain enough understanding to follow the work presented in this dissertation. However, this is not intended to be a rigorous mathematical treatment of power laws (there are several papers that give a much more comprehensive explanation [116] [88] [118] [159] [160]).

Generally, when working with power laws, the rare events go in the far right of the distribution's tail. This convention causes problems when dealing with different types of data. Here I will use a comparison between word frequency data [110] [111] and firm size data [122], which both follow a power law. In word frequency data discussed in Chapter 4, the word 'the' is the most common word and the word 'macaws' is the $100,000^{th}$ most common word. Therefore, 'the' is used frequently, while 'macaws' is not, so 'macaws' belongs in the tail of the distribution. However, if we consider firm sizes, there is currently a single publicly held company with over 2 million employees (Walmart), while there are many small companies with only a single employee. Therefore, using the convention that rarity goes in the tail, Walmart should go in the tail of the distribution. This creates the discrepancy. In the word frequency case, the entity with the first rank is very common and in the firm size case, is very rare. There seems to be a consistent availability bias [205] at play. This occurs because there are many instances of the word 'the', but only one Walmart. For this dissertation, I have followed convention and placed the rare events in the tail, which requires different interpretations of the power law.

Now that the reason for the discrepancy is explained, I will briefly explain the two formulations of power laws used in this dissertation: Zipf and Pareto (this discussion is based on material from [206] [118] and [125]). The confusion with the two interpretations is further intensify by the common usage of α for the exponents in both types of power laws. To avoid this confusion, I will use α_z for Zipf's formulation and α_p for Pareto's formulation.

For this dissertation, I use α_p in Chapter 2 and α_z in Chapter 4.

A power law follows:

$$P(x) \propto x^{-\alpha} \tag{E.1}$$

This means that the probability of some event, $P(x)$, is proportional to the size of the event to some power, $-\alpha$, where α must be greater than or equal to 1. A power law in terms of Zipf's formulation is:

$$P(r) \propto r^{-\alpha_z} \tag{E.2}$$

Where r is the rank of the event ordered by decreasing frequency. Zipf's formulation can also be written as:

$$f(r) = \frac{C}{r^{\alpha_z}} \tag{E.3}$$

Now, $f(r)$ is the frequency of rank r and C is the frequency of the most common item (which means that $C = f(1)$). If the logarithm is taken of Equation E.3, then the terms can be rearranged to be the same form as a linear equation ($y = mx + b$):

$$\log f(r) = -\alpha_z \log r + \log C \tag{E.4}$$

This shows how if you plot the power law on a log-log plot, then it will be a straight line with slope of $-\alpha_z$.

Zipf's formulation plots a power law with the rank (r) on the x-axis and the frequency ($f(r)$) on the y-axis. However, Pareto's formulation has the two axes swapped. Now, the frequency goes on the x-axis, but the interpretation of the y-axis changes a bit. Saying that an entity has rank r is equivalent to saying that the entity is smaller than $r - 1$ other entities. So, to place this in terms of a cumulative density function[1]:

$$P(X \geq f(r)) = \frac{r}{N} \tag{E.5}$$

[1]Equation E.5 assumes that ranks i and $i+1$ do not have the same frequency, which is reasonable because the Pareto distribution is continuous.

Where $P(X \geq f(r))$ is the probability that a random sample drawn from the distribution is greater than or equal to the frequency of the entity with rank r. N is the total number of entities in the distribution. By using Equation E.5 to solve for r and then combining with Equation E.3, we get:

$$f(r) = \frac{C}{(P(X \geq f(r))N)^{\alpha_z}} \tag{E.6}$$

By taking the logarithm of this equation and rearranging terms to follow the linear equation ($y = mx + b$):

$$\log P(X \geq f(r)) = -\frac{1}{\alpha_z} \log f(r) - \log(N) + \frac{\log(C)}{\alpha_z} \tag{E.7}$$

As noted earlier, the axes have been switched so the $P(X \geq f(r))$ goes along the y-axis and the $f(r)$ term along the x-axis. Therefore, on a log-log plot, the slope of this line is $-\frac{1}{\alpha_z}$. But, this is in regards to the cumulative density function and Pareto's formulation is generally stated in terms of the probability density function. So, after taking the derivative of Equation E.7 to obtain the probability density function, the slope becomes $-\left(\frac{1}{\alpha_z} + 1\right)$. Finally, ranks are not used in Pareto's formulation, so x replaces $f(r)$ and we arrive to Pareto's Law:

$$P(X = x) \propto x^{-\left(\frac{1}{\alpha_z}+1\right)} = x^{-\alpha_p} \tag{E.8}$$

To summarize, going by convention, α_p is used in Chapter 2 and α_z in Chapter 4 and you can switch from one interpretation to the other using this equation:

$$\alpha_p = \frac{1}{\alpha_z} + 1 \tag{E.9}$$

Appendix F: Technologies Used

This section will give a brief overview of the various tools and software libraries that were used in this dissertation. MATLAB and R was used throughout this dissertation. Several R libraries were found to be quite helpful, including: stats, stats4, graphics, DAAG, MASS, VGAM, and DistributionUtils. Some of the analyses of the programming contest data and the agent-based model were written in Java. The agent-based model used the MASON simulation library[1]. Some of the custom visualizations were created with Processing[2]. The symbolic regression was performed using Eureqa Formulize[3].

This dissertation required much computational time and memory to complete. Various machines were used, including: two MacBook Pros (one with 4GB of memory and the other with 8GB), a Mac workstation (with 64GB of memory), and a Linux machine running Fedora (with 16GB of memory).

[1]For more information about MASON: http://cs.gmu.edu/ eclab/projects/mason/
[2]For more information about Processing: http://processing.org/
[3]For more information about Eureqa Formulize: http://formulize.nutonian.com/

Bibliography

Bibliography

[1] E. D. Beinhocker, *The origin of wealth: Evolution, complexity, and the radical remaking of economics*. Harvard Business school Press, 2006.

[2] G. Grossman and E. Helpman, *Innovation and growth in the global economy*. MIT press, 1993.

[3] N. Gilbert and K. Troitzsch, *Simulation for the Social Scientist*, 2nd ed. Open University Press, 2005.

[4] G. N. Gilbert, *Agent-based models*. Los Angeles: Sage Publications, 2008.

[5] L. Georghiou, "Global cooperation in research," *Research Policy*, vol. 27, no. 6, pp. 611–626, 1998.

[6] C. Wagner and L. Leydesdorff, "Network structure, self-organization, and the growth of international collaboration in science," *Research policy*, vol. 34, no. 10, pp. 1608–1618, 2005.

[7] G. Walt, "Globalisation of international health," *The Lancet*, vol. 351, no. 9100, pp. 434–437, 1998.

[8] D. Heymann and G. Rodier, "Sars: A global response to an international threat," *Brown J. World Aff.*, vol. 10, p. 185, 2003.

[9] C. Fraser, C. Donnelly, S. Cauchemez, W. Hanage, M. Van Kerkhove, T. Hollingsworth, J. Griffin, R. Baggaley, H. Jenkins, E. Lyons *et al.*, "Pandemic potential of a strain of influenza A (H1N1): early findings," *Science*, vol. 324, no. 5934, pp. 1557–1561, 2009.

[10] J. Sebenius, "Designing negotiations toward a new regime: The case of global warming," *International Security*, vol. 15, no. 4, pp. 110–148, 1991.

[11] J. Corfee-Morlot, M. Maslin, and J. Burgess, "Global warming in the public sphere," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 365, no. 1860, pp. 2741–2776, 2007.

[12] A. Newell and H. Simon, *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.

[13] W. B. Arthur, *The Nature of Technology: What it is and how it evolves*. New York: Free Press, 2009.

[14] K. De Jong, *Evolutionary computation: a unified approach.* The MIT Press, 2006.

[15] T. Back, D. Fogel, and Z. Michalewicz, *Handbook of evolutionary computation.* IOP Publishing Ltd., 1997.

[16] K. Sawyer, *Group Genius: The Creative Power of Collaboration.* Basic Books, Mar. 2008.

[17] "MATLAB on-line programming contest home page." [Online]. Available: http://www.mathworks.com/contest/

[18] "Google books," http://books.google.com/ngrams.

[19] J. Surowiecki, *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations.* Doubleday, 2004.

[20] R. Florida, *The Rise of the Creative Class: And How It's Transforming Work, Leisure, Community and Everyday Life.* Basic Books, 2003.

[21] S. Page, *The Difference: How the Power of Diversity Creates Better Groups, Firms, Schools, and Societies.* Princeton University Press, 2007.

[22] ——, *Diversity and Complexity*, 1st ed. Princeton University Press, 2011.

[23] R. Ogle, *Smart World: Breakthrough Creativity And the New Science of Ideas*, 1st ed. Harvard Business School Press, 2007.

[24] M. Csikszentmihalyi, *Flow: the psychology of optimal experience.* Harper & Row, 1990.

[25] G. Basalla, *The Evolution of Technology.* Cambridge, UK: Cambridge University Press, 1988.

[26] K. Kelly, *What Technology Wants.* Viking Adult, 2010.

[27] T. S. Kuhn, *The Structure of Scientific Revolutions*, 3rd ed. Chicago: University Of Chicago Press, Dec. 1996.

[28] J. A. Schumpeter, *The Theory of Economic Development, Translated from the German by Redvers Opie.* Harvard University Press, 1934.

[29] H. Simon, *The Sciences of the Artificial - 3rd Edition.* The MIT Press, 1996.

[30] S. Gould, *The Structure of Evolutionary Theory*, 1st ed. Belknap Press of Harvard University Press, 2002.

[31] E. M. Rogers, *Diffusion of Innovations.* New York: Free Press, 1962.

[32] C. M. Christensen, "Exploring the limits of the technology S-curve. part I: Component technologies," *Production and Operations Management*, vol. 1, No. 4, pp. 334–357, 1992.

[33] J. A. Schumpeter, *Capitalism, Socialism and Democracy.* New York: Harper and Row, 1942.

[34] J. Schmookler, *Invention and Economic Growth.* Cambridge, MA: Harvard University Press, 1966.

[35] D. Sahal, *Patterns of Technological Innovation.* London: Addison-Wesley, 1981.

[36] H. Koh and C. L. Magee, "A functional approach for studying technological progress: Application to information technology," *Technological Forecasting and Social Change*, vol. 73, pp. 1061–1083, 2006.

[37] R. Kurzweil, *The Singularity Is Near: When Humans Transcend Biology*, 1st ed. Viking Adult, 2005.

[38] C. Cioffi-Revilla, "A canonical theory of origins and development of social complexity," *Journal of Mathematical Sociology*, vol. 29, no. 2, pp. 133–153, 2005.

[39] N. Eldredge and S. J. Gould, "Punctuated equilibria: An alternative to phyletic gradualism," *Production and Operations Management*, vol. 1, No. 4, pp. 334–357, 1992.

[40] W. Abernathy and J. Utterback, "Patterns of industrial innovation," *Technology Review*, vol. 80, no. 7, pp. 40–47, 1978.

[41] J. Utterback and F. Suarez, "Innovation, competition, and industry structure," *Research policy*, vol. 22, no. 1, pp. 1–21, 1993.

[42] J. Schmookler, *Patents, Invention, and Economic Change.* Cambridge, MA: Harvard University Press, 1972.

[43] G. Dosi, "Technological paradigms and technological trajectories: a suggested interpretation of the determinants and directions of technical change," *Research policy*, vol. 11, no. 3, pp. 147–162, 1982.

[44] G. Di Stefano, A. Gambardella, and G. Verona, "Technology push and demand pull perspectives in innovation studies: Current findings and future research directions," *Research Policy*, vol. 41, pp. 1283–1295, 2012.

[45] D. Mowery and N. Rosenberg, "The influence of market demand upon innovation: a critical review of some recent empirical studies," *Research Policy*, vol. 8, no. 2, pp. 102–153, 1979.

[46] R. R. Nelson and S. G. Winter, *An Evolutionary Theory of Economic Change.* Cambridge, MA: The Belknap Press of Harvard University Press, 1982.

[47] K. Frenken, *Innovation, Evolution and Complexity Theory.* Cheltenham, UK: Edward Elgar Publishing, 2006.

[48] R. Solow, "Technical change and the aggregate production function," *The Review of Economics and Statistics*, vol. 39, pp. 312–320, August 1957.

[49] S. Kauffman, *The Origins of Order: Self-Organization and Selection in Evolution.* New York and Oxford: Oxford University Press, 1993.

[50] L. Altenberg, "Evolving better representations through selective genome growth," *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 182–7, 1994.

[51] ——, "Genome growth and the evolution of the genotype-phenotype map," *Evolution and Biocomputation*, pp. 205–59, 1995.

[52] ——, "Nk fitness landscapes," *The Handbook of Evolutionary Computation*, 1997.

[53] O. Williamson, *Markets and Hierarchies.* New York: Free Press, 1975.

[54] J. Cairns, J. Overbaugh, and S. Miller, "The origin of mutants," *Nature*, vol. 335, no. 6186, pp. 142–145, 1988.

[55] S. Kauffman, *The Evolution of Economic Webs.* Reading, MA: Addison-Wesley, 1988, vol. The Economy as an Evolving Complex System, pp. 125–146.

[56] P. Auerswald, S. Kauffman, J. Lobo, and K. Shell, "The production recipes approach to modeling technological innovation: An application to learning by doing," *Journal of Economic Dynamics and Control*, vol. 24, pp. 389–450, March 2000.

[57] K. Arrow, "The economic implications of learning by doing," *The Review of Economic Studies*, vol. 29, no. 3, pp. 155–173, 1962.

[58] T. P. Wright, "Factors Affecting the Cost of Airplanes," *Journal of the Aeronautical Sciences*, vol. 2, pp. 122–128, 1936.

[59] D. Gode and S. Sunder, "Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality," *The Journal of Political Economy*, vol. 101, no. 1, pp. 119–137, 1993.

[60] ——, "What makes markets allocationally efficient?" *The Quarterly Journal of Economics*, vol. 112, no. 2, pp. 603–630, 1997.

[61] H. Simon, "Rationality in psychology and economics," *The Journal of Business*, vol. 59, no. 4, pp. S209–S224, 1986.

[62] ——, "A behavioral model of rational choice," *The Quarterly Journal of Economics*, vol. 69, no. 1, pp. 99–118, 1955.

[63] ——, "Invariants of human behavior." *Annual review of psychology*, vol. 41, no. 1, pp. 1–19, 1990.

[64] B. Edmonds, "Modeling bounded rationality in agent-based simulations using the evolution of mental models," *Computational Techniques for Modeling Learning in Economics, Advances in Computational Economics*, pp. 305–32, 1999.

[65] B. Ebersberger and A. Pyka, "The use of genetic programming in evolutionary economics," in *Applied evolutionary economics and complex systems.* Edward Elgar, 2004, pp. 78–94.

[66] S.-H. Chen, "On the relevance of genetic programming to evolutionary economics," in *Evolutionary Controversies in Economics: A New Transdisciplinary Approach*. Tokyo: Spring-Verlag, 2001.

[67] R. Lucas, "Adaptive behavior and economic theory," in *Rational choice: the contrast between economics and psychology*, R. Hogarth and M. Reder, Eds. University of Chicago Press, 1986, pp. 217–242.

[68] S.-H. Chen, "Genetic programming and agent-based computational economics: From autonomous agents to product innovation," in *Agent-Based Approaches in Economic and Social Complex Systems V*, T. Terano, H. Kita, S. Takahashi, and H. Deguchi, Eds. Springer, 2007.

[69] S.-H. Chen and B.-T. Chie, "A functional modularity approach to agent-based modeling of the evolution of technology," in *Lecture Notes in Economics and Mathematical Systems, The Complex Networks of Economic Interactions*, A. Namatame, T. Kaizouji, and Y. Aruka, Eds., 2006, vol. 567, pp. 165–178.

[70] J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, T. G. B. Team, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, and E. L. Aiden, "Quantitative analysis of culture using millions of digitized books," *Science*, vol. 331, no. 6014, pp. 176–182, 2011. [Online]. Available: http://www.sciencemag.org/content/331/6014/176.abstract

[71] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge, MA: MIT press, 1992.

[72] ——, "Building blocks, cohort genetic algorithms, and hyperplane-defined functions," *Evol. Comput.*, vol. 8, no. 4, pp. 373–391, Dec. 2000. [Online]. Available: http://dx.doi.org/10.1162/106365600568220

[73] R. Axtell, *Why agents? : on the varied motivations for agent computing in the social sciences*. Washington DC; Baltimore MD: Center on Social and Economic Dynamics, November 2000.

[74] K. Clark, "The interaction of design hierarchies and market concepts in technological evolution," *Research Policy*, vol. 14, pp. 235–51, 1985.

[75] W. B. Arthur, "Why do things become more complex?." *Scientific American*, vol. 268, no. 5, p. 144, 1993.

[76] M. Harrower and C. Brewer, "Colorbrewer.org: An online tool for selecting colour schemes for maps," *Cartographic Journal, The*, pp. 27–37, 2003.

[77] C. A. Brewer, "Guidelines for use of the perceptual dimensions of color for mapping and visualization," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, Bares, J., Ed., vol. 2171, no. 1, May 1994, pp. 54–63.

[78] ——, "Color use guidelines for mapping and visualization," *Visualization in modern cartography*, vol. 2, pp. 123–148, 1994.

[79] R. M. Casstevens, "What leads to innovation: An analysis of data from software developers." Presented at the 2nd Annual Complexity in Business Conference, Washington, DC, 2010.

[80] ——, "What leads to innovation: An analysis of collaborative problem-solving." Presented at the 7th European Meeting on Applied Evolutionary Economics, Pisa, Italy, 2011.

[81] N. Gulley, "Patterns of innovation: A web-based MATLAB programming contest," *Extended Abstracts of CHI 2001*, pp. 337–338, March-April 2001.

[82] ——, "In praise of tweaking: a wiki-like programming contest," *Interactions*, vol. 11, pp. 18–23, May + June 2004 2004.

[83] N. Gulley and K. Lakhani, "The determinants of individual performance and collective value in private-collective software innovation," *Harvard Business School Technology & Operations Mgt. Unit Working Paper*, no. 10-065, 2010.

[84] F. P. Brooks, Jr., *The Mythical Man-Month.* Boston, MA: Addison-Wesley, 1995.

[85] G. Madey, V. Freeh, and R. Tynan, "Understanding OSS as a self-organizing process," *The 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering (ICSE2002)*, 2002.

[86] T. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.

[87] I. D. Baxter, A. Yahin, L. M. D. Moura, M. Sant'anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees," in *International Conference on Software Maintenance*, 1998, pp. 368–377.

[88] A. Clauset, C. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009.

[89] P. Good, *Permutation Tests: A Practical Guide to Resampling Methods for Testing Hypotheses*, 2nd ed. Springer, 2000.

[90] J. Maindonald and J. Braun, *Data analysis and graphics using R : an example-based approach*, 3rd ed. Cambridge University Press, 2010.

[91] E. D. Weinberger., "NP completeness of kauffman's n-k model, a tuneable rugged fitness landscape tuneable rugged fitness landscape." Santa Fe Institute, Santa Fe, NM, Tech. Rep. 96-02-003, 1996. [Online]. Available: http://www.santafe.edu/sfi/publications/96wplist.html

[92] A. Wright, R. Thompson, and J. Zhang, "The computational complexity of N-K fitness functions," *Evolutionary Computation, IEEE Transactions on*, vol. 4, no. 4, pp. 373–379, 2000.

[93] S. Forrest, "Genetic algorithms: principles of natural selection applied to computation," *Science*, vol. 261, pp. 872–878, 1993.

[94] K. A. De Jong, M. A. Potter, and W. M. Spears, "Using Problem Generators to Explore the Effects of Epistasis," in *International Conference on Genetic Algorithms*, 1997, pp. 338–345.

[95] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley Professional, 1989.

[96] J. Holland, *Hidden Order: How Adaptation Builds Complexity (Helix Books)*. Addison Wesley Publishing Company, 1998.

[97] D. Kahneman, *Thinking, fast and slow*. Allen Lane, 2011.

[98] M. Schmidt and H. Lipson, "Distilling Free-Form Natural Laws from Experimental Data," *Science*, vol. 324, pp. 81–85, 2009.

[99] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge: MIT Press, 1992.

[100] ——, *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge: MIT Press, 1994.

[101] P. Wagstrom, J. Herbsleb, and K. Carley, "A social network approach to free/open source software simulation," in *First International Conference on Open Source Systems*, 2005, pp. 16–23.

[102] Z. Michalewicz and D. Fogel, *How to Solve It: Modern Heuristics*. Springer, 2004.

[103] K. R. Popper, *All Life Is Problem Solving*. Routledge, 1999.

[104] R. Ferrer i Cancho and R. Solé, "Two regimes in the frequency of words and the origins of complex lexicons: Zipfs law revisited¡sup¿*¡/sup¿," *Journal of Quantitative Linguistics*, vol. 8, no. 3, pp. 165–173, 2001.

[105] M. Montemurro, "Beyond the zipf-mandelbrot law in quantitative linguistics," *Physica A Statistical Mechanics and its Applications*, vol. 300, pp. 567–578, 2001.

[106] M. H. R. Stanley, L. A. N. Amaral, S. V. Buldyrev, S. Havlin, H. Leschhorn, P. Maass, M. A. Salinger, and H. E. Stanley, "Scaling behaviour in the growth of companies," *Nature*, vol. 379, pp. 804–806, 1996.

[107] G. Minnen, J. Carroll, and D. Pearce, "Robust, applied morphological generation," in *Proceedings of the first international conference on Natural language generation-Volume 14*. Association for Computational Linguistics, 2000, pp. 201–208.

[108] ——, "Applied morphological processing of English," *Natural Language Engineering*, vol. 7, 2001.

[109] A. Spencer, *Morphological theory: An introduction to word structure in generative grammar*. Basil Blackwell, 1991, vol. 2.

[110] G. Zipf, *The psycho-biology of language: an introduction to dynamic philology*. Boston, MA: Houghton Mifflin, 1935.

[111] ——, *Human Behaviour and the Principle of Least Effort: an Introduction to Human Ecology*. Cambridge, MA: Addison-Wesley, 1949.

[112] B. Mandelbrot, "An informational theory of the statistical structure of language," *Communication theory*, pp. 486–502, 1953.

[113] G. Yule, *The statistical study of literary vocabulary*. CUP Archive, 1944.

[114] P. Juola, "Authorship attribution," *Foundations and Trends® in Information Retrieval*, vol. 1, no. 3, pp. 233–334, 2007.

[115] D. Holmes, "The evolution of stylometry in humanities scholarship," *Literary and linguistic computing*, vol. 13, no. 3, pp. 111–117, 1998.

[116] M. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005.

[117] J. Estoup, *Gammes sténographiques: méthode et exercices pour l'acquisition de la vitesse*. Institut sténographique, 1916.

[118] W. Li, "Zipf's Law Everywhere," *Glottometrics*, vol. 5, pp. 14–21, 2002.

[119] V. Pareto, *Cours d'economie politique*, 1896.

[120] B. Hill, "Zipf"s Law and Prior Distribution for the Composition of a Population," *Journal of the American Statistical Association*, vol. 65, pp. 1220–1232, 1970.

[121] K. T. Rosen and M. Resnick, "The size distribution of cites: An examination of the pareto law and primacy," *Journal of Urban Economics*, vol. 8, pp. 165–186, September 1980.

[122] R. L. Axtell, "Zipf distribution of u.s. firm sizes," *Science*, vol. 293, no. 5536, pp. 1818–1820, September 2001.

[123] A. J. Lotka, "The frequency distribution of scientific productivity," *Journal of the Washington Academy of Sciences*, vol. 16, pp. 317–323, 1926.

[124] D. J. D. S. Price, "Networks of scientific papers," *Science*, vol. 149, no. 3683, pp. 510–515, July 1965.

[125] L. A. Adamic and B. A. Huberman, "Zipf's law and the Internet," *Glottometrics*, vol. 3, pp. 143–150, 2002.

[126] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM '99)*, pp. 126–134, 1999.

[127] R. Albert, H. Jeong, and A. L. Barabasi, "The diameter of the world wide web," *Nature*, vol. 401, pp. 130–131, 1999.

[128] B. Huberman and L. Adamic, "Growth dynamics of the world wide web," *Nature*, vol. 401, no. 6749, p. 131, 1999.

[129] J. Voss, "Measuring wikipedia," *Proceedings of the 10th International Conference of the International Society for Scientometrics and Informetrics (ISSI)*, pp. 221–231, July 2005.

[130] M. Small and J. Singer, *Resort to arms: international and civil wars, 1816-1980.* Sage Publications, Inc, 1982.

[131] D. Roberts and D. Turcotte, "Fractality and self-organized criticality of wars," *Fractals*, vol. 6, no. 04, pp. 351–357, 1998.

[132] S. Pinker, "Survival of the clearest," *Nature*, vol. 404, pp. 441–442, 2000.

[133] A. Perfors, "Simulated Evolution of Language: a Review of the Field," *The Journal of Artificial Societies and Social Simulation*, vol. 5, 2002.

[134] A. Tsonis, C. Schultz, and P. Tsonis, "Zipf's law and the structure and evolution of languages," *Complexity*, vol. 2, no. 5, pp. 12–13, 1999.

[135] V. S. Grishchenko. (2006, November) Back to the basics: Zipf's law. [Online]. Available: http://bouillon.math.usu.ru/index.html%3Fp=79.html

[136] J. Tuldava, "The frequency spectrum of text and vocabulary," *Journal of Quantitative Linguistics*, vol. 3, no. 1, pp. 38–50, 1996.

[137] H. Situngkir, "Regimes in babel are confirmed: Report on findings in several indonesian ethnic biblical texts," *Available at SSRN 984102*, 2007.

[138] G. Miller, "Some effects of intermittent silence," *The American Journal of Psychology*, pp. 311–314, 1957.

[139] B. Mandelbrot, "On the theory of word frequencies and on related markovian models of discourse," *Structure of language and its mathematical aspects*, pp. 190–219, 1961.

[140] ——, *The fractal geometry of nature.* Times Books, 1982.

[141] W. Li, "Random texts exhibit zipf's-law-like word frequency distribution," *Information Theory, IEEE Transactions on*, vol. 38, no. 6, pp. 1842–1845, 1992.

[142] R. Perline, "Zipf's law, the central limit theorem, and the random division of the unit interval," *Physical Review E*, vol. 54, no. 1, p. 220, 1996.

[143] M. Schroeder, *Fractals, chaos, power laws: Minutes from an infinite paradise.* Dover Publications, 2009.

[144] R. Ferrer i Cancho and R. Solé, "Zipf's law and random texts," *Advances in Complex Systems*, vol. 5, no. 01, pp. 1–6, 2002.

[145] V. Balasubrahmanyan and S. Naranan, "Quantitative linguistics and complex system studies*," *Journal of Quantitative Linguistics*, vol. 3, no. 3, pp. 177–228, 1996.

[146] S. Naranan and V. Balasubrahmanyan, "Information theoretic models in statistical linguistics. ii: Word frequencies and hierarchical structure in language-statistical tests," *Current science*, vol. 63, no. 6, pp. 297–306, 1992.

[147] G. Wimmer, R. Köhler, R. Grotjahn, and G. Altmann, "Towards a theory of word length distribution*," *Journal of Quantitative Linguistics*, vol. 1, no. 1, pp. 98–106, 1994.

[148] A. Cohen, R. Mantegna, and S. Havlin, "Numerical analysis of word frequencies in artificial and natural language texts," *Fractals*, vol. 5, no. 01, pp. 95–104, 1997.

[149] U. Yule, "A mathematical theory of evolution, based on the conclusions of dr. j. c. willis, f.r.s." *Philosophical Transactions of the Royal Society of London. Series B, Containing Papers of a Biological Character*, vol. 213, no. 402-410, pp. 21–87, 1925.

[150] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[151] X. Gan, D. Wang, and Z. Han, "A growth model that generates n-tuple zipf law," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 5, pp. 792–800, 2010.

[152] A. Czirók, H. Stanley, and T. Vicsek, "Possible origin of power-law behavior in n-tuple zipf analysis," *Physical Review E*, vol. 53, no. 6, p. 6371, 1996.

[153] A. Czirók, R. Mantegna, S. Havlin, and H. Stanley, "Correlations in binary sequences and a generalized zipf analysis," *Physical Review E*, vol. 52, no. 1, p. 446, 1995.

[154] R. Mantegna, S. Buldyrev, A. Goldberger, S. Havlin, C. Peng, M. Simons, and H. Stanley, "Linguistic features of noncoding dna sequences," *Physical review letters*, vol. 73, no. 23, pp. 3169–3172, 1994.

[155] W. Dahui, L. Menghui, and D. Zengru, "True reason for zipf's law in language," *Physica A: Statistical Mechanics and its Applications*, vol. 358, no. 2-4, pp. 545–550, 2005.

[156] L. Q. Ha, E. I. Sicilia-garcia, J. Ming, and F. J. Smith, "Extension of zipf's law to word and character n-grams for english and chinese," in *Journal of Computational Linguistics and Chinese Language Processing*, 2003, pp. 77–102.

[157] D. Zanette and M. Montemurro, "Dynamics of text generation with realistic zipf's distribution," *Journal of Quantitative Linguistics*, vol. 12, no. 1, pp. 29–40, 2005.

[158] C. Tsallis, G. Bemski, and R. Mendes, "Is re-association in folded proteins a case of nonextensivity?" *Physics Letters A*, vol. 257, no. 1, pp. 93–98, 1999.

[159] S. Baek, S. Bernhardsson, and P. Minnhagen, "Zipf's law unzipped," *New Journal of Physics*, vol. 13, no. 4, p. 043004, 2011.

[160] L. Adamic, "Complex systems: Unzipping zipf's law," *Nature*, vol. 474, no. 7350, pp. 164–165, 2011.

[161] R. Albert and A. Barabási, "Topology of evolving networks: local events and universality," *Physical review letters*, vol. 85, no. 24, pp. 5234–5237, 2000.

[162] D. Gay, "Usage summary for selected optimization routines," *Computing Science Technical Report*, vol. 153, 1990.

[163] H. Akaike, "A new look at the statistical model identification," *Automatic Control, IEEE Transactions on*, vol. 19, no. 6, pp. 716–723, 1974.

[164] G. Schwarz, "Estimating the dimension of a model," *The annals of statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[165] B. Mandelbrot, "Structure formelle des textes et communication," *Word*, vol. 10, pp. 1–27, 1954.

[166] ——, *Information theory and psycholinguistics: A theory of word frequencies*, P. Lazarsfeld and N. Henry, Eds. Chicago: Science Research Associates Inc., 1966.

[167] I. Kanter and D. Kessler, "Markov processes: linguistics and zipf's law," *Physical review letters*, vol. 74, no. 22, pp. 4559–4562, 1995.

[168] R. Ferrer i Cancho and R. Solé, "The small world of human language," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, vol. 268, no. 1482, pp. 2261–2265, 2001.

[169] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[170] S. Dorogovtsev and J. Mendes, "Language as an evolving word web," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, vol. 268, no. 1485, pp. 2603–2606, 2001.

[171] ——, "Scaling behaviour of developing and decaying networks," *EPL (Europhysics Letters)*, vol. 52, no. 1, p. 33, 2000.

[172] N. Chomsky, *Remarks on nominalization*. Waltham: Ginn, 1968, vol. Reading in English Transformational Grammar, pp. 184–221.

[173] R. Jackendorff, "X-bar-syntax: A study of phrase structure," *Linguistic Inquiry Monograph*, vol. 2, 1977.

[174] N. Chomsky, *Syntactic structures*. de Gruyter Mouton, 2002.

[175] ——, *Aspects of the Theory of Syntax*. MIT press, 1969, vol. 119.

[176] ——, "Approaching UG from below," *Interfaces+ recursion= language*, pp. 1–29, 2007.

[177] C. Beckner, R. Blythe, J. Bybee, M. Christiansen, W. Croft, N. Ellis, J. Holland, J. Ke, D. Larsen-Freeman, and T. Schoenemann, "Language is a complex adaptive system: Position paper," *Language Learning*, vol. 59, no. s1, pp. 1–26, 2009.

[178] S. Pinker, *The language instinct: How the mind creates language*. Harper Perennial Modern Classics, 2007.

[179] K. Toutanova and C. Manning, "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger," in *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*. Association for Computational Linguistics, 2000, pp. 63–70.

[180] K. Toutanova, D. Klein, C. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, 2003, pp. 173–180.

[181] D. Fu, F. Pammolli, S. Buldyrev, M. Riccaboni, K. Matia, K. Yamasaki, and H. Stanley, "The growth of business firms: Theoretical framework and empirical evidence," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 52, pp. 18 801–18 806, 2005.

[182] M. Wyart and J. Bouchaud, "Statistical models for company growth," *Physica A: Statistical Mechanics and its Applications*, vol. 326, no. 1, pp. 241–255, 2003.

[183] M. Riccaboni, F. Pammolli, S. Buldyrev, L. Ponta, and H. Stanley, "The size variance relationship of business firm growth rates," *Proceedings of the National Academy of Sciences*, vol. 105, no. 50, pp. 19 595–19 600, 2008.

[184] A. Petersen, J. Tenenbaum, S. Havlin, and H. Stanley, "Statistical laws governing fluctuations in word use from word birth to word death," *Scientific Reports*, vol. 2, 2012.

[185] R. A. Blythe, "Neutral evolution: A null model for language dynamics," *Advances in Complex Systems*, vol. 15, no. 03n04, 2012.

[186] M. Beckmann, "City hierarchies and the distribution of city size," *Economic Development and Cultural Change*, vol. 6, no. 3, pp. 243–248, 1958.

[187] W. Hsu, "Central place theory and zipf's law," *Working Paper, University of Minnesota*, 2008.

[188] R. Axtell and R. Casstevens, "Innovation by purposive agents produces coevolution of economic goods and increasing technological complexity," in *World Congress on Social Simulation*, 2008.

[189] "The SourceForge research data archive (SRDA)," 2008. [Online]. Available: http://zerlot.cse.nd.edu/

[190] M. Van Antwerp and G. Madey, "Advances in the sourceforge research data archive (srda)," in *Fourth International Conference on Open Source Systems, IFIP 2.13 (WoPDaSD 2008)*, Milan, Italy, September 2008.

[191] "SourceForge," 2008. [Online]. Available: http://www.sourceforge.net

[192] Y. Gao and G. Madey, "Towards understanding: A study of the sourceforge.net community using modeling and simulation," *Agent-Directed Simulation (ADS'07) - Part of the 2007 Spring Simulation Multiconference (SpringSim'07), The Society for Modeling and Simulation International (SCS)*, pp. 145–150, March 2007.

[193] G. Madey, V. Freeh, R. Tynan, Y. Gao, and C. Hoffman, "Agent-based modeling and simulation of collaborative social networks," *Americas Conference on Information Systems (AMCIS2003)*, August 2003.

[194] S. Christley and G. Madey, "Global and temporal analysis of social positions at sourceforge.net," *The Third International Conference on Open Source Systems (OSS 2007), IFIP WG 2.13*, June 2007.

[195] J. Xu and G. Madey, "Exploration of the open source software community," *NAACSOS*, June 2004.

[196] J.-M. Dalle and P. A. David, "Simcode: Agent-based simulation modelling of open-source software development," EconWPA, Industrial Organization, 2005. [Online]. Available: http://econpapers.repec.org/RePEc:wpa:wuwpio:0502008

[197] J. Moon and L. Sproull, "Essence of distributed work," *Online Communication and Collaboration: A Reader*, p. 125, 2010.

[198] L. Torvalds, "The linux edge," *Communications of the ACM*, vol. 42, no. 4, pp. 38–39, 1999.

[199] S. Weber, *The success of open source*. Cambridge Univ Press, 2004, vol. 368.

[200] N. P. Radtke and M. A. Janssen, "Consumption and production of digital public goods: Modeling the impact of different success metrics in open source software development," *International Journal of Intelligent Control and Systems*, vol. 14, no. 1, pp. 77–86, March 2009.

[201] N. P. Radtke, M. A. Janssen, and J. S. Collofello, "What makes free/libre open source software (FLOSS) projects successful? an agent-based model of FLOSS projects," *International Journal of Open Source Software and Processes*, vol. 1, no. 2, pp. 1–13, April-June 2009.

[202] L. Torvalds and D. Diamond, *Just for fun: The story of an accidental revolutionary*. HarperBusiness, 2002.

[203] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel," *Research policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[204] R. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[205] A. Tversky and D. Kahneman, "Availability: A heuristic for judging frequency and probability," *Cognitive Psychology*, vol. 5, no. 2, pp. 207–232, 1973.

[206] L. A. Adamic, "Zipf, power-law, pareto - a ranking tutorial," http://www.hpl.hp.com/research/idl/papers/ranking/, 2000.

Curriculum Vitae

Randy Casstevens studied Computer Science at North Carolina State University where he received Bachelor of Science and Master of Science degrees. While a student at N.C. State, Randy had several jobs within the university, including: undergraduate teaching assistant, tutor, supplemental instruction leader, graduate research assistant and disc jockey for the university's radio station. He also took advantage of the cooperative education program through the university where he gained experience working for IBM, BellSouth Telecommunications, and Westinghouse Electric Corporation. After leaving N.C. State, Randy worked as a teaching associate for the University of Massachusetts, a software engineer for the MIT Lincoln Laboratory and a consultant for Charles River Analytics. He became interested in using his computer science skills to study social science problems. Thus, he joined the Department of Computational Social Science at George Mason University as a Presidential Scholar. While getting a Ph.D. in Computational Social Science, his research focused on the evolution of technology, and specifically on how building blocks are recombined to create innovations. As a Ph.D. student, he presented his work at conferences in various disciplines, including: social simulation, business, and evolutionary economics.