THE EVOLUTIONARY POTENTIAL OF POPULATIONS ON COMPLEX FITNESS LANDSCAPES

By

David Michael Bryson

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science
Ecology, Evolutionary Biology and Behavior

2012

ABSTRACT

THE EVOLUTIONARY POTENTIAL OF POPULATIONS ON COMPLEX FITNESS LANDSCAPES

By

David Michael Bryson

Evolution is a highly contingent process, where the quality of the solutions produced is affected by many factors. I explore and describe the contributions of three such aspects that influence overall evolutionary potential: the prior history of a population, the type and frequency of mutations that the organisms are subject to, and the composition of the underlying genetic hardware. I have systematically tested changes to a digital evolution system, Avida, measuring evolutionary potential in seven different computational environments ranging in complexity of the underlying fitness landscapes. I have examined trends and general principles that these measurements demonstrate and used my results to optimize the evolutionary potential of the system, broadly enhancing performance. The results of this work show that history and mutation rate play significant roles in evolutionary potential, but the final fitness levels of populations are remarkably stable to substantial changes in the genetic hardware and a broad range of mutation types.

Very special thanks to
Emmalena J. Gregory-Bryson for love and support
Elena S. Bryson for helpful distraction

In memory of
Unnamed Bryson
8/6/2012

ACKNOWLEDGMENT

I would like to extend my appreciation to the members of my guidance committee, Richard Lenski, Erik Goodman, Eric Torng, Robert Pennock, and especially my advisor and committee chair, Charles Ofria. Their collaboration, support, and feedback has been invaluable. Thanks to Zachary Blount, Arthur Covert III, Heather Goldsby, Matthew Rupp, Aaron Wagner, and other members of the Michigan State University Digital Evolution Laboratory for helpful discussion and critique. Special thanks to Jeffrey Barrick for the inspiration, initial implementation, and collaboration on the early studies of slip mutations.

This material is based in part upon work supported by the National Science Foundation under Cooperative Agreement No. DBI-0939454 and Grant CCF-0643952 Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Evolution by natural selection is a powerful process, having produced the incredible diversity and complexity observed in the natural world. Biological evolution appears to demonstrate seemingly limitless evolutionary potential, capable of continuously adapting to the environment. Still, while evolution may be broadly capable on such a grand scale, individual microcosms appear subject to the vagaries of history (Blount *et al.*, 2008). Beyond basic research questions, an understanding of how specific details of evolving systems affect evolutionary potential could have important implications for a wide range of challenges in evolutionary biology, virology, bacteriology, immunology, oncology, and others. For example, given an understanding of the factors that constrain evolutionary potential, we may be able to limit pathogen evolution and prevent host shifts, or to develop antibiotics that simultaneously reduce or prevent the evolution of resistance.

The field of evolutionary computation seeks to harness the power of evolution to solve difficult engineering problems. These techniques have shown great promise and have advanced our capability to approach complex challenges Koza (2003). However, remaining limitations

to evolution in the computational realm include a capability for open-ended evolution of complexity, as observed in nature. Evolutionary computation systems eventually stagnate, failing to achieve further improvements, despite their theoretical availability. Understanding factors that determine evolutionary potential is necessary if we are to overcome this limitation. A great deal of research has focused on increasing potential, such as preventing diversity loss or improving fitness functions and mutational operators.

There are many aspects of an evolving system that may limit or enhance its evolutionary potential, including the prior history of a population, the type and frequency of mutations that the organisms are subject to, and the composition of the underlying genetic hardware. While there are other aspects that could be examined, these are the three that I selected to explore systematically.

The prior history of an evolving population encompasses all of its historical events and conditions, including environmental pressures, competition dynamics, and random chance. These factors leave a lasting signature on the heritable substrate in an evolving system, which determines where the organisms reside in a complex fitness landscape. Different regions of a fitness landscape may have dramatically different properties, including epistatic interactions between sites, robustness to mutations, and adaptive potential. Taken altogether, the prior history of a population may play a major role in its evolutionary potential.

Mutations are a fundamental source of variation in evolving systems, defining the types and frequency of changes that can be made to the heritable substrate, and hence the possible trajectories that can be taken across the fitness landscape of a given environment. The resulting rules governing traversal can make some regions of a landscape more or less accessible, directly impacting evolutionary potential. Mutations also influence the degree to which

epistasis can be overcome, the organization and architecture of the functional characters in the heritable substrate, and the robustness of evolved instances – all important aspects that can limit or enhance evolutionary potential.

Here, the genetic hardware defines the characters of the heritable substrate and the semantics of their interactions. The genetic hardware is the underlying foundation upon which all aspects of evolution by natural selection ultimately act. In natural organisms, the genetic hardware encompasses DNA, RNA, cellular machinery, and the biochemistry that governs function. The genetic hardware defines the potential functional capabilities of individuals in the system, *how* the characters encoded in the heritable substrate can be changed, and the underlying architecture of the genetic hardware defines *what* those changes will actually do. The genetic hardware establishes the rules by which a genotype is translated into the actual phenotype upon which selection acts. The interactions of components of the genetic hardware, at the genotype level and in the translation of it, heavily influence the necessary epistasis of evolved instances, defining the strength of connections required between components and how robust or brittle they are to changes. Due to the foundational nature of these properties, I expect the choice of genetic hardware to play a critical role in shaping the fitness landscape and ultimately the evolutionary potential of an evolving system as a whole.

Digital evolution, which encompasses the essentials of the evolutionary process in a controlled, flexible, and transparent computational environment, is a powerful tool for studying evolving systems (Ofria *et al.*, 2009). I have used digital evolution to better understand how the above aspects of evolving systems affect evolutionary potential at scopes and scales that would have been intractable, if not impossible, with natural systems. I have measured and

described the consequences of history, investigating how the internal state of a population impacts the future evolutionary potential of that population. In my exploration of mutations and genetic hardware I have taken a top-down approach, altering the characteristics of the system and identifying how to enhance the evolutionary potential of the system in the context of each.

The insight gained from these experiments will have significant impact on the design of future digital evolution systems. I have identified general design principles, parameter selection guidelines, and novel techniques that support enhanced systematic evolutionary potential in the form of greater population mean fitness and long-term task success across multiple environments. Furthermore, the knowledge gained provides a broad understanding of how these core aspects of all evolving systems shape evolutionary potential. This understanding will allow researchers to make more accurate predictions about the evolutionary potential of a system.

Chapter 2

Methods

I have performed all experiments using executables based on the Avida 2.x source code.[1] Avida is a computational platform for performing experiments on evolving populations of digital organisms. An experimental world in Avida consists of a two-dimensional grid of cells that may each contain a single organism. Except where noted, experiments use worlds with 60-by-60 cells, supporting a maximum population size of 3,600 organisms. A digital organism in this virtual world contains a central processing unit (CPU) that executes a genomic program of assembly-like instructions based on a configured instruction set architecture. The genomic program is responsible for directing organism behavior, including interactions with the environment and replication. In all experiments described here, offspring organisms are placed into the local neighborhood, the nine cells immediately connected to, and including, the cell of the parent organism.

Organisms that replicate in Avida are subject to mutation. These mutations can take many forms, including substitution of one instruction for another (point mutations), instruc-

[1]The Avida Git source code repository may be retrieved from http://avida.devosoft.org

tion insertion, and instruction deletion, among others. Except where otherwise specified, organisms were subject to a substitution mutation rate of 2.5×10^{-3} per site in the genome, along with a 5×10^{-4} probability each for a single insertion or deletion per site in the genome. The standard substitutions, insertions, and deletions occur during the division of the offspring, after the parent organism has completed the copy process.

I seeded all cells in initial populations with identical organisms, all sharing a hand-written ancestral genotype capable only of self replication. This ancestral genotype contains three sections in its genomic program: (1) setup instructions that prepare for replication, (2) a series of non-functional `nop-C` instructions, that take time and can modify other instruction behavior, but do not otherwise alter organism behavior when executed, and (3) a simple loop-based replication algorithm. The overall length of the ancestral genotype was either 50 or 100 instructions, varying only in the number of `nop-C` instructions between the header and the replication loop (32 or 82, respectively). The replication algorithm copies the genomic program one instruction at a time from the source to a subsequent region of memory. The ancestral genome ends with a specific marker template that the organism can identify when it has been copied. The default replication algorithm uses this signal to trigger offspring division, extracting the region of memory containing the copied program as the genome of the offspring. Once replication is complete, the state of the virtual CPU is completely reset, including all memory, registers, stacks, buffers, and head locations.

2.1 The HEADS Instruction Set Architecture

The HEADS instruction set architecture in Avida is the default virtual CPU configuration.[2] The virtual hardware that implements this instruction set is designed to operate on a genomic program within a circular memory space, such that operations that reach the end of the memory automatically wrap around to the beginning (as shown on the left side of Figure 2.1. By default, it has three 32-bit registers, two stacks that can each hold ten values, four heads that point to positions in the genome, input and output buffers, and the ability to execute 26 standard instructions. The default instructions include three no-operation instructions (nops): (`nop-A`, `nop-B`, and `nop-C`), which can serve to modify the default behavior of other instructions. Most instructions observe the value of one subsequent nop instruction and alter their behaviors accordingly. For instance, an instruction may change the source registers used as operands when followed by a nop instruction. In addition to instruction modification, nop instructions can serve as patterns, which act as labels for genome locations. Label matching uses cyclic complementary matching, where `nop-A` matches to `nop-B`, `nop-B` matches to `nop-C`, and `nop-C` matches to `nop-A`.

The HEADS instruction set has five flow-control instructions: `h-search`, `jmp-head`, `mov-head`, `get-head`, and `set-flow`. Each of these instructions can affect the position of one of the four architectural heads: the instruction pointer (IP), READ head, WRITE head, and FLOW head. The `h-search` instruction searches the genome, starting from the first executed instruction in the genome, for a label (a sequence of one or more nop instructions) that matches the cyclic complementary label that follows the instruction, placing the FLOW

[2] All versions of Avida 2.x through version 2.13 share a functionally equivalent default virtual CPU configuration. Some specific details in the configuration files have changed across versions.

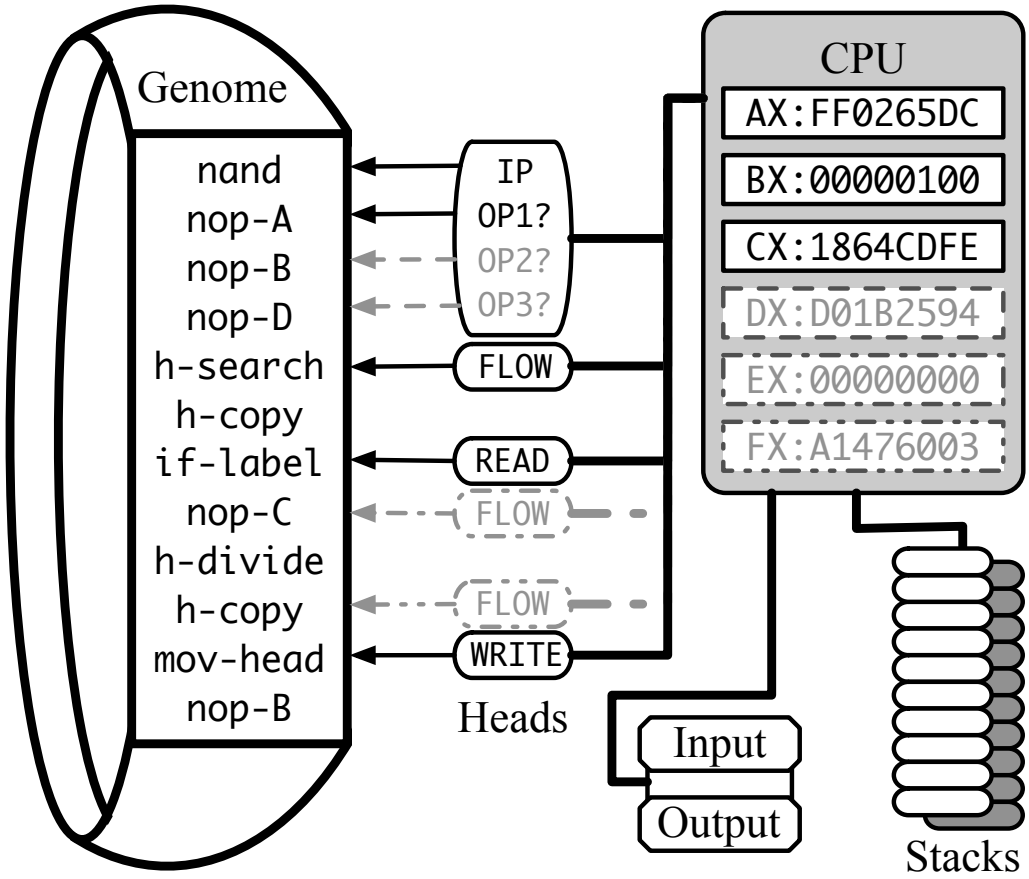

Figure 2.1: The architecture of the Avida virtual CPU
Registers (upper right), stacks (lower right), genomic program (left), heads (middle), and environmental buffers (lower right). The solid lines depict the default HEADS architectural features. The dashed lines show some of the modifications tested in Chapter 4.

head after the matching sequence; if the sought-after label is not found, it places the FLOW head on the instruction immediately subsequent to itself. This is one of only two instructions in the default HEADS instruction set that is affected by more than one nop instruction, the other being `if-copied` described below. The `mov-head` instruction moves the IP to the current location of the FLOW head. The `jmp-head` instruction shifts the position of the IP by the amount specified in a register. The `get-head` instruction places the current location of the IP into a register. Finally, the `set-flow` instruction moves the FLOW head to the absolute genome location specified by the value in a register.

The set also contains three conditional instructions that will skip a subsequent instruction if the test condition is false. The two basic conditional instructions, `if-n-equ` and `if-less`, perform a comparison between two registers. The `if-copied` instruction interacts with the READ head, evaluating to true if the last sequence of instructions copied matches the complement of the label that follows the instruction. This instruction is primarily for use in conjunction with the replication instructions described below to identify the portion of the genome most recently copied.

Seven arithmetic and logic operations are supported in the default HEADS instruction set: `add`, `sub`, `inc`, `dec`, `nand`, `shift-l`, and `shift-r`. All of these instructions operate on values stored within registers and accept a single nop modifier, which changes the source and destination registers depending on the operation.

Five instructions facilitate data movement and environmental interaction. The `push`, `pop`, and `swap-stk` instructions all operate on the two stacks within the architecture. Only one stack is accessible at a time, with the swap-stk instruction toggling the currently active stack, while `push` and `pop` copy numbers from registers to the top of the stack and vice-versa. Each of these instructions can be nop-modified to specify which register should be used. The `swap` instruction exchanges the values of two registers. The `IO` instruction interacts with the environment of the digital organism, outputting the current value in a register and replacing it with a value from the environmentally controlled input buffer. Values output via this instruction are evaluated by the environment, potentially triggering a reward or other action if they match one of the tasks in the environment as explained below.

Lastly, there are three instructions that facilitate self-replication. The `h-alloc` instruction allocates additional memory within which the digital organism can copy its offspring.

Environment	Rewarded Functions
Logic-9	Calculate nine logic operations with one or two inputs.
Logic-77	Calculate 77 logic operations with one, two, or three inputs.
Match-12	Generate up to 12 specific numbers.
Fibonacci-32	Output up to 32 numbers of the Fibonnaci sequence, in order.
Sort-10	Input 10 random numbers and output in correctly sorted order.
Limited-9	Logic-9 environment with a limited resource associated with each task.
Navigation	Successfully traverse a labeled pathway.

Table 2.1: The seven environments used to test evolutionary potential.

Copying is performed by repeated execution of the `h-copy` instruction, which duplicates the current instruction found at the READ head to the position marked by the WRITE head and advances both heads. Once copying has been completed, the organism must execute the `h-divide` instruction to finalize the replication process, extracting the memory between the READ head and the WRITE head as the genome of the offspring.

2.2 Environments

Avida supports a wide range of computational environments. For the work presented here, I have used seven distinct environments (Table 2.1), each of which focuses on a different aspect of the virtual architecture and presents unique evolutionary pressures. Activities (or tasks) whose performance provide a metabolic reward define the environment. These rewards increase the computation speed of the digital organism's virtual CPU, making it possible to obtain a competitive advantage relative to other organisms in the population.

2.2.1 Logic-9

The *Logic-9* environment consists of metabolic rewards for all possible 1- and 2-input binary logic operations; there are nine unique operations after removing symmetries and excluding trivial operations. The environment rewards the performance of these tasks multiplicatively, thus virtual CPU speed will increase exponentially as the organism performs additional tasks. There are five reward levels associated with groups of logic operations, ranked by difficulty. The easiest group (NOT and NAND) will double computational speed, while the highest level (EQU) increases execution speed by thirty-two times. The environment rewards each task only once during an organism's lifetime. This environmental setup is the default for Avida and many previous experiments have used it (Lenski *et al.*, 1999, 2003; Misevic *et al.*, 2006).

2.2.2 Logic-77

The *Logic-77* environment increases the size and complexity of the Logic-9 environment by adding a reward for all unique 3-input binary logic operations. In contrast to the Logic-9 environment, all operations provide an equal benefit, doubling the execution speed of the virtual CPU for the first time the organism performs computation. Yedid *et al.* (2009) used this environment.

2.2.3 Match-12

The *Match-12* environment tests organisms' ability to build arbitrary numbers, a task that we have previously observed to constitute an obstacle to evolution (unpublished data). The environment grants rewards in an additive manner for outputting each of twelve possible

numbers, unrelated to the random inputs. We selected numbers spaced approximately exponentially throughout the 32-bit number space, but the numbers contain no explicit patterns to them. The environment rewards the output of each number only once during an organism's lifetime. Output evaluation allows near matches, but the reward decays via a half-life function based upon the number of bits that are incorrect with a minimum threshold of 22 bits correct to prevent most numbers from triggering many 'lucky guesses'.

2.2.4 Fibonacci-32

The *Fibonacci-32* environment rewards organisms multiplicatively for each number in the Fibonacci sequence until the 32nd iteration of the sequence. After this target, the environment penalizes the organism at half this rate for additional numbers output, whereby outputting 64 additional numbers will effectively negate all benefit of the first 32. The purpose of this setup is to examine the capability of an instruction set to support finite recursion and conditional looping.

2.2.5 Sort-10

The *Sort-10* environment supplies a list of 10 random inputs, and rewards organisms for outputting those values in descending order. Similar to the Match-12 environment, the reward value decays via a half-life function for each incorrectly sorted value, based on the number of moves required to shift it to the correct order. Given the limited number of available registers, this task requires the use of the stacks and relatively complex flow control.

2.2.6 Limited Resource

The *Limited-9* environment, based on the Logic-9 environment, offers metabolic rewards each of the nine rewarded logic operations. However, unlike the Logic-9 environment, the Limited-9 environment associates a separate, consumable resource with each task, the amount of which determines the exact reward value. Each resource flows into the environment at a rate of 100 units per update, and out at 1% of the remaining concentration. If no organisms are using the resource it will level out to $10,000$ units. This environment was first used in Cooper and Ofria (2003) and was recently used in Walker and Ofria (2012).

2.2.7 Navigation

The *Navigation* environment rewards organisms for successfully navigating a circuitous path marked by cues ("sign posts") including "turn left", "turn right", and "repeat last turn", as described in Grabowski *et al.* (2010). This task requires the use of basic memory, looping, and decision making. Additionally, the environment tests robustness of instruction set architectures to the addition of several experiment-specific instructions, in this instance for sensing and moving in the virtual maze.

2.3 Analysis Tools and Methods

In order to conduct the analysis of the experiments performed, as described below, I required several tools, many of which I developed for this purpose. Here I describe the general operation of these tools, reserving the details of their application for the experiment methods sections below.

2.3.1 Mutational Neighborhood

Mutational neighborhood analysis is the systematic exploration of potential mutant genotypes surrounding a reference genotype. This method examines all possible single and double mutations for all classes of mutations present in a given experimental setup. Each mutant genotype is evaluated for fitness, task profile (phenotype), and an information-theoretic measure of complexity. I also collected aggregate statistics regarding fitness of the mutant genotype relative to the reference genotype, shifts in phenotype, and changes in complexity. These data provide a wealth of information regarding the local fitness landscape of the reference genotype.

The mutational neighborhood tool currently implements this method for all possible combinations of single and double mutations from three types of mutation: single-instruction substitution (point mutation), single-instruction insertion, and single-instruction deletion. Additional information is collected for genotypes demonstrating the acquisition of a selected target task, including counts of mutation pathways that confer the target task, along with mean relative fitness of all such pathways discovered. When two-step pathways that confer the selected target task are identified, the class of each mutation taken individually in that pathway is also tracked, be it beneficial, neutral, deleterious, or lethal.

2.3.2 Short-Term Evolutionary Potential (STEP) Sampling

Short-term evolutionary potential (STEP) sampling explores the mid-range fitness landscape of a reference genotype by evolving repeated short runs from the same starting point and analyzing aggregate statistics of the outcome of each. This procedure involves injecting the reference genotype as a single organism in an otherwise empty experimental world. I

then allow the world to evolve for a short period, 10,000 updates (approximately 1,000 generations) for the work presented here, after which I collect metrics of interest, such as phenotype and fitness. I repeat this procedure with the same reference genotype multiple times for statistical assessment of the genotype's evolutionary potential (10 for all studies presented here).

2.3.3 Data Analysis and Statistical Testing

Data analyses and statistical tests were performed using MATLAB R2012a, Python 2.7.3, Matplotlib 1.10, NumPy 1.6.2, SciPy 0.10.1, MySQLdb 1.2.3, MySQL 5.5.23, and the Apto C++ Library (pre-release 1.0).

Chapter 3

How Does History Affect Evolutionary Potential?

The evolutionary process shapes the genomes of organisms through a combination of both chance and determinism (Mayr, 1988). Random mutations introduce novel genetic variation that can affect phenotypic traits while natural selection acts non-randomly on such heritable phenotypic variation (Conner and Hartl, 2004). Despite the strength of natural selection, genetic drift may randomly cull adaptive mutants before selection can fix them (Conner and Hartl, 2004).

Individual sites within a genome and indeed whole genes in organisms can form highly interactive networks to express a phenotype. As a result of these epistatic interconnections, the fitness of a mutation can depend upon the genetic background in which it occurs. Fixation of a mutation yields a new background, altering the fitness and likelihood of future mutations (Weinreich and Chao, 2005; Weinreich *et al.*, 2006). The order in which mutations occur can therefore take genotypes along different mutational pathways through the

adaptive landscape. Such alternate paths may lead to different fitness peaks (Wright, 1988).

Steven J. Gould has argued that, in the context of such properties, chance and historical events make evolution prone to historical contingency and therefore fundamentally unpredictable (Gould, 1989). Contingency is the property of history whereby outcomes follow from antecedent states, both unpredictably and causally dependent upon prior states (Beatty, 2006). To explain this, Gould posed a thought experiment:

> I call this experiment "replaying life's tape." You press the rewind button and, making sure you thoroughly erase everything that actually happened, go back to any time and place in the past.... Then let the tape run again and see if the repetition looks at all like the original (Gould, 1989).

He expected that any replay of the tape would lead evolution down a pathway radically different from the road actually taken (Gould, 1989).

Others suggest that more striking is the importance of determinism and thus predictability. The pervasiveness of parallel and convergent evolution support this assertion, as seen in the multiple independent development of eyes, echolocation, and social systems, among other innovations (Conway Morris, 2003; Dawkins, 1996b,a). Common challenges will inevitably arise from ecological interactions, for which the scope of viable solutions may be limited. Faced with such sparse options, natural selection is robust enough to discover these solutions as a lineage adapts, regardless of its history (Conway Morris, 2003; Dawkins, 1996b,a). Although important differences would result from any single replay of the tape, multiple replays would also demonstrate notable similarity (Van Valen, 1991). As Simon Conway Morris puts it "the evolutionary routes are many, but the destinations are limited" (Conway Morris, 2003).

In the field of evolutionary computation, it is just as important to understand the role

of history in evolution as it is in biology. Genetic algorithms (Holland, 1975), genetic programming (Koza and Poli, 2005), evolutionary programming (Fogel, 1995), and evolutionary strategies (Rechenberg, 1971) all seek to efficiently solve computational and engineering problems through the use of evolutionary principles. Contingent, historical events may constrain the search space of a given evolutionary algorithm run, limiting the effectiveness of the search. Given the size and complexity of target problems and the high cost of evaluating potential solutions, it is crucial to maximize search efficiency (Eby *et al.*, 1999).

As evolutionary outcomes are defined by both random events and non-random processes, the challenge is to define the degree to which contingency shapes those outcomes (Foote, 1998). A small but growing number of studies have sought to provide such insight, looking into the relative roles of chance, history, and adaptation in evolution, as well as contingency in macroevolutionary trends (Blount *et al.*, 2008; Travisano *et al.*, 1995; Taylor and Hallam, 1998; Losos *et al.*, 1998; Yedid and Bell, 2002). In light of this work, I have explored the role of history in determining evolutionary potential and to gain an understanding of how historical events shape the trajectory of evolution. The studies presented below focus on the evolution of the EQU task from the Logic-9 environment, a trait that has been shown in previous work to be complex and contingent upon building-block tasks (Lenski *et al.*, 2003). First, I performed a Gould-inspired replay experiment, investigating the contribution of history to the evolution of EQU. Following that, I used two means of measuring evolutionary potential along the principal line of descent of several case study populations and assessed the long-term impact of historical mutations. Lastly, I examine the characteristics of stepping-stone mutations that bring genotypes within regions of the fitness landscape close to the evolution of the EQU function.

3.1 Assessing the Contribution of History

Stephen J. Gould's thought experiment of "replaying life's tape", though impossible on the grand scale he envisioned, has the power to demonstrate the relative contributions of history, chance, and determinism. Using digital evolution, we can take historical snapshots, from which we can restart evolution. With this capability, it becomes possible to truly replay the tape of digital life, examining the repeatability of outcomes. Moreover, this replay experiment can itself be replicated with multiple independent histories, allowing conclusions to be drawn about the specific contribution of history to the evolutionary potential of the subsequently repeated replays.

3.1.1 Methods

I evolved a total of 100 independent populations for 100,000 updates using the standard configuration described in Chapter 2. The environment used for these 100 runs was a variant of the Logic-9 environment — one in which performance of the complex EQU task did not provide a metabolic reward. I saved the complete genotypic history as well as the distribution and age of all organisms at the end of the 100,000 updates for all 100 populations.

I used the saved population information from each run to found 10 new replicate populations each, for a total of 1,000 replicates grouped as 100 shared-history clusters. I placed these populations into the standard Logic-9 environment (including reward for performing EQU) and allowed them to evolve for 900,000 additional updates. Due to the shared history from the initial 100 runs, each of the 1,000 replicates experienced a total of 10^6 updates of evolution from the ancestral population, 10% of which was shared with 9 other replicates. I examined the task profile of the most abundant (dominant) genotype from the end of each

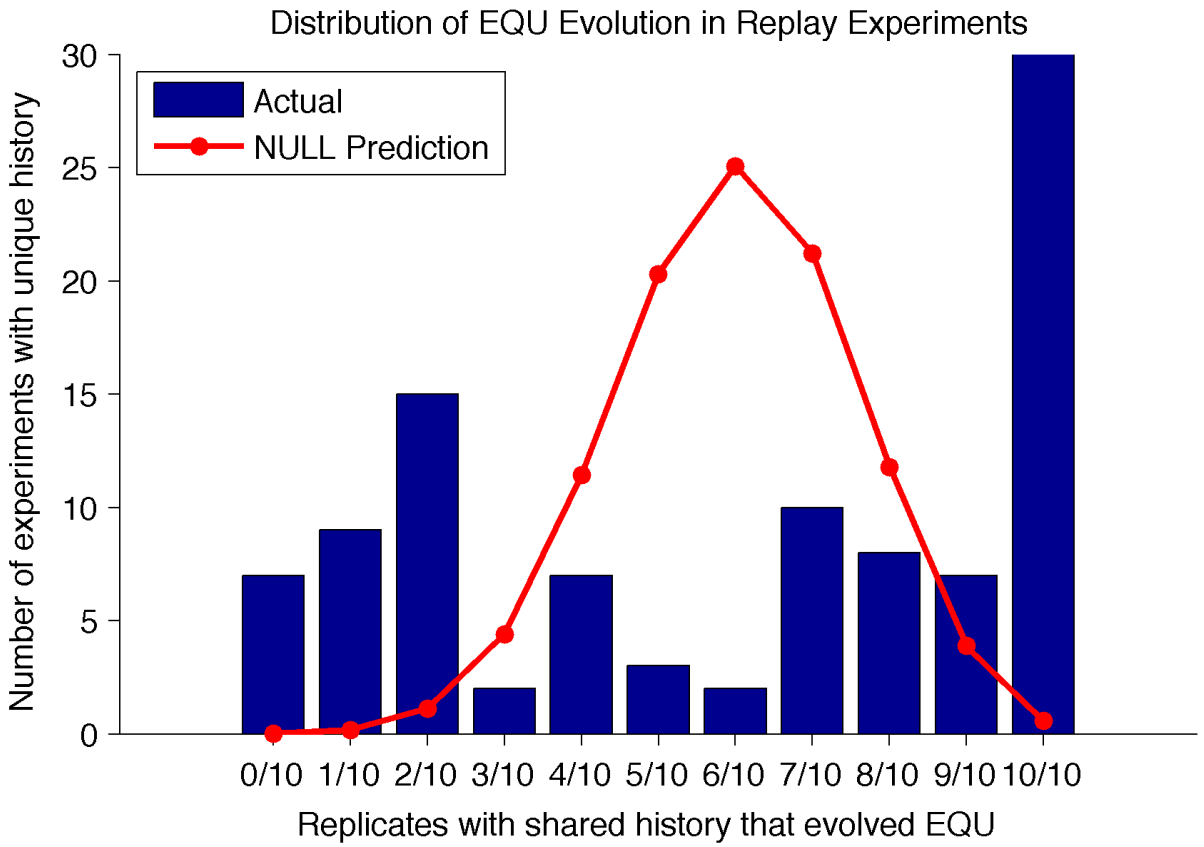

Figure 3.1: Distribution of EQU Evolution in Replay Experiments
Distribution of EQU predicted by a null model (red line) and experimentally observed (blue bars). Each bar represents the number of shared-history clusters where EQU evolved the specified number of times. The null model is the expected binomial distribution that should have been seen if shared history played no role, and thus there was no correlation between runs in a shared-history cluster. For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this dissertation.

of the 1,000 replicate populations for EQU performance.

3.1.2 Results

A total of 597 populations evolved to calculate the EQU function at the end of the 1,000 replays performed. The resulting populations performing EQU were organized into shared-history clusters. Among these shared-history clusters, thirty demonstrated 10/10 replays

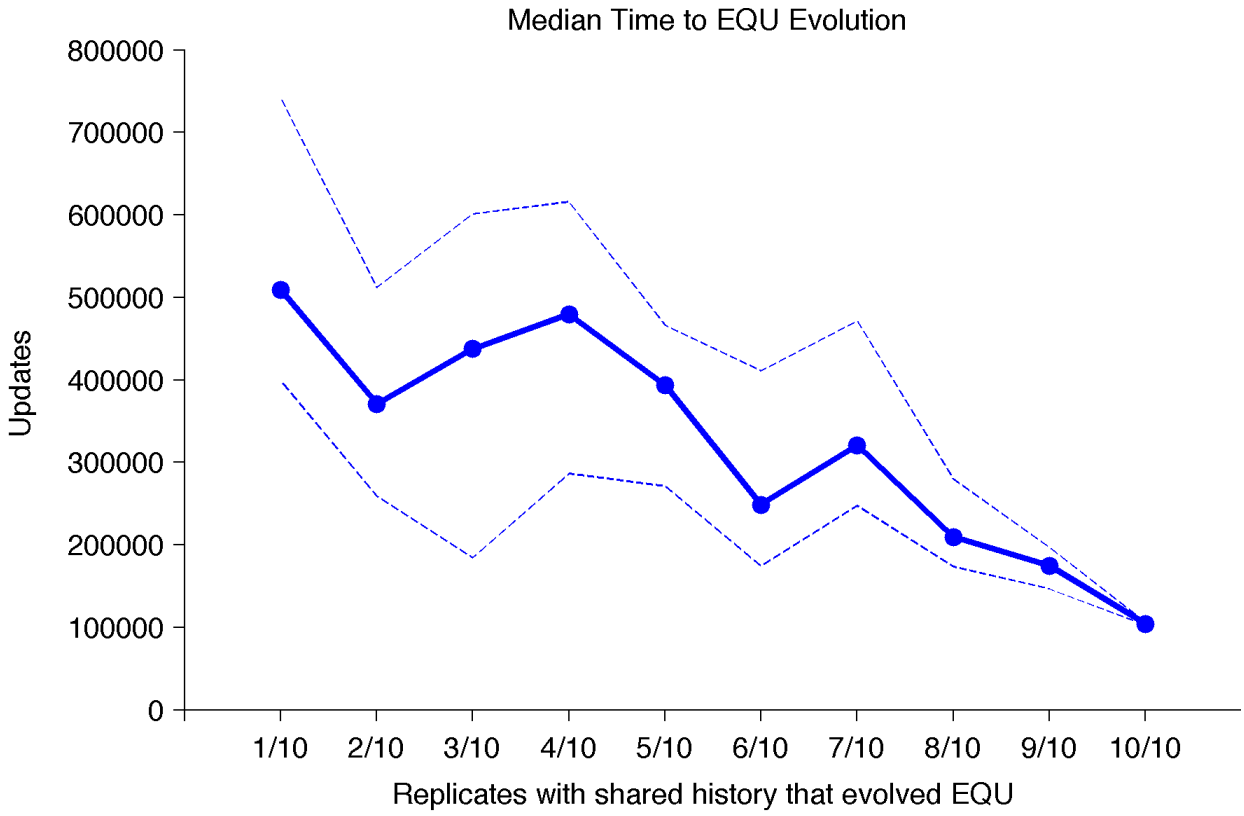

Figure 3.2: Median time to EQU evolution
Each point represents the value observed for shared-history clusters where EQU evolved the specified number of times. The y-axis shows total updates of evolution, including the 100,000 updates from shared history. Dotted lines show the 95% non-parametric bootstrap confidence intervals (10,000 iterations).

performing the function, seven of them 9/10, eight of them 8/10, etc. The full distribution of results can be found in Figure 3.1. For comparison, if history were not a factor in the evolution of EQU, the distribution of populations would be expected to follow a binomial distribution with $p = 0.597$, also shown in Figure 3.1. The observed results significantly diverge from the binomial expectation ($p < 5.084 \times 10^{-20}$, Fisher's Exact).

Overall, the median time to evolve the EQU function in successful populations was 135,291 total updates, including the 100,000 updates of history shared among replay groups, with 95% non-parametric bootstrap confidence interval of 123,340 to 152,944. When the

times to evolve EQU were grouped according to shared history and fraction of successful replays, a notable trend was observed (Figure 3.2). Historical populations that resulted in 10/10 replays evolving the EQU function had a median time to EQU of only 104,336 updates (102,769, 105,750.5), significantly lower the global median ($p < 3.174 \times 10^{-23}$, Wilcoxon rank sum test). At the opposite end of the spectrum, historical populations that yielded only one successful replay demonstrated median time to EQU of 509,512 updates (398,881, 742,679), significantly higher than both the 10/10 populations ($p < 1.025 \times 10^{-6}$) and the overall median ($p < 9.980 \times 10^{-5}$).

3.1.3 Discussion

History clearly plays a substantial role in determining evolutionary potential of the EQU function. In comparison to the null expectation that history has no impact on future evolution, the observed results are strikingly divergent. The variation in time to evolve EQU between groups of shared history presents hints at potential explanations for the strong effect of history. As shared-history clusters demonstrated less aggregate success, the median time to reach the observed instances of EQU evolution also increased. The history of these less successful populations may have taken the genotypes to regions of the fitness landscape farther away from the evolution of EQU than their more successful counterparts. Constraints, such as epistasis and genetic substrate, may be slowing these populations down and limiting overall evolutionary potential with respect EQU.

3.2 Measuring Change in Evolutionary Potential

History substantially contributes to the future evolutionary potential of populations with respect to the evolution of the EQU function. Implicit in that conclusion is that there are some aspects of the historical genotypes that enhance or limit the ability of future generations to evolve a particular phenotypic trait, in this case EQU. Evolution in some of those histories traversed into regions of the fitness landscape that made it highly unlikely for the function to evolve in the next 900,000 updates. In other histories, the landscape presented great potential for evolving EQU, with the rest filling out the spectrum of possibilities. These observations raise questions about how evolutionary potential changes during evolution. Are there particular types of mutations that decrease the potential for EQU evolution, and if so, are those mutations typically beneficial when they come into the genotype? As genotypes approach regions of the fitness landscape close to the EQU function, what do the mutation pathways to the task look like? Does EQU typically require steps through fitness valleys, contributing to its rarity? In the experiments described here, I attempt to answer these questions by directly exploring the fitness landscape surrounding genotypes throughout the history of selected runs. I have applied two techniques to exploring the landscape, exhaustive mutational neighborhood mapping and STEP sampling.

3.2.1 Methods

I evolved fifty reference populations for 10^6 updates in the Logic-9 environment and collected historical genotype data. At the end of each experiment I extracted the principal line of descent, defined as the complete lineage from the ancestral genotype to the most abundant genotype at the end of the run. I used a test environment to evaluate the task profile of each

genotype along each of these principal lines of decent. I truncated all lineages at the first genotype capable performing EQU if one was found. I then used these truncated-EQU and non-EQU lineages to perform the two analysis methods described in the following sections.

In order to aid the analysis of the experiments, the study populations evaluated in this section utilize a configuration that differs in a number of ways from the description in the methods chapter. The most important among these differences is the mutations that were experienced by the organisms: rather than the standard substitution, insertion, and deletion mutations, the organisms were subject to a uniform mutation that chose equally among 53 possibilities, 26 instruction substitutions, 26 instruction insertions, or a deletion. Additionally, a standard length-based adjustment factor that is typically applied to metabolic rate was disabled, increasing selection pressure for the evolution of more rapid production of offspring. The changes allowed for direct interpretation of mutation probabilities calculated during mutational neighborhood analysis.

3.2.2 Mutational Neighborhood Lineage Analysis

I performed mutational neighborhood analysis for all genotypes along each principal line of descent (PLoD) extracted from the fifty populations. All possible one- and two-step mutation pathways conferring the non-lethal[1] performance of the EQU task were identified. I further classified two-step mutation pathways according to the fitness effect of each mutation in the path, providing an indication of the character of the pathway if the mutations were taken individually, rather than combined. Although the individual results of the 50 lineages all

[1]It is possible for a genotype to be capable of computing the EQU function, while being incapable of producing an offspring. Mutations that generate such genotypes were classified as 'lethal' and excluded from identified pathways.

Group	# Pops	Found EQU?	When near EQU in genotype space?
I	11	Yes	Just prior to EQU appearing
II	6	Yes	Sporadically throughout lineage
III	6	Some	Most of lineage
IV	4	No	Sporadically throughout lineage
V	3	Yes	Early followed by long period away, returning just before EQU appeared
VI	2	No	Early, but never returned
VII	4	No	Long contiguous region
VIII	14	No	Rarely or never

Table 3.1: General groups observed during mutational neighborhood analysis
Each row shows the group number, number of populations that fit into the group, and the defining characteristic of the group.

had unique features (see Appendix A), I observed eight general types of traversals through the fitness landscape (Table 3.1).

Group I, representing 11 of the populations examined, demonstrated a reasonably short, mostly contiguous section of the lineage where the genotype was within two mutations of the evolution of EQU before the task actually appeared in the line of descent. The two-step pathways in these regions varied in the quality of their individual steps, showing different distributions of beneficial, neutral, deleterious, and lethal steps. Almost universally the pathways contain some deleterious single steps. The pathways in this group frequently showed neutral pathways as well, and occasionally some extremely rare beneficial steps. Figure 3.3 shows a representative example of results from this group, where the EQU task appeared in the genotype immediately following the block of two-step pathways. As can be observed in the figure, there were also single-step pathways that non-lethally conferred the EQU task.

Group II, representing 6 populations, demonstrated similar patterns leading up to the

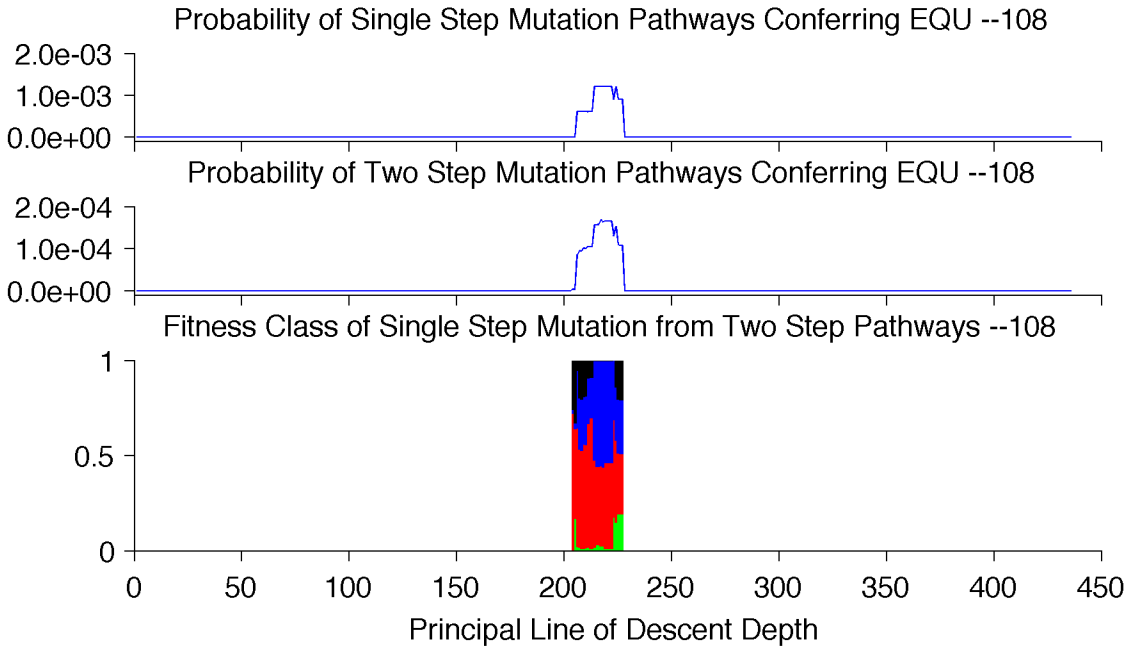

Figure 3.3: Mutational Neighborhood Group I Example — Population 108
Mutational neighborhood of the PLoD from population 108, representative of Group I. The top graph shows the absolute probability of a single mutation conferring performance of EQU. The middle graph shows the absolute probability of two mutations occurring simultaneously that, combined, confer performance of EQU. Note that the top and middle graphs are on distinct scales. The bottom graph shows the quality of individual mutations within two-step pathways when taken alone, grouped into four general classes: beneficial (green), neutral (blue), deleterious (red), lethal (black).

evolution of EQU. However, unlike the first group, these populations also included genotypes at numerous points earlier in the line of descent that were within two mutations of the task, but never found it at those points. These earlier two-step pathways generally had low probability of occurring, with each mutation typically being deleterious or lethal if taken alone. Figure 3.4 shows an example population from this group.

The six lines of descent from Group III were even more extreme: they contained genotypes that were within two mutations of EQU for most of their histories, but either mutation singly would have been deleterious or lethal, although on rare occasion some neutral paths were available. Only one of the six populations eventually evolved EQU, population 109 (Figure

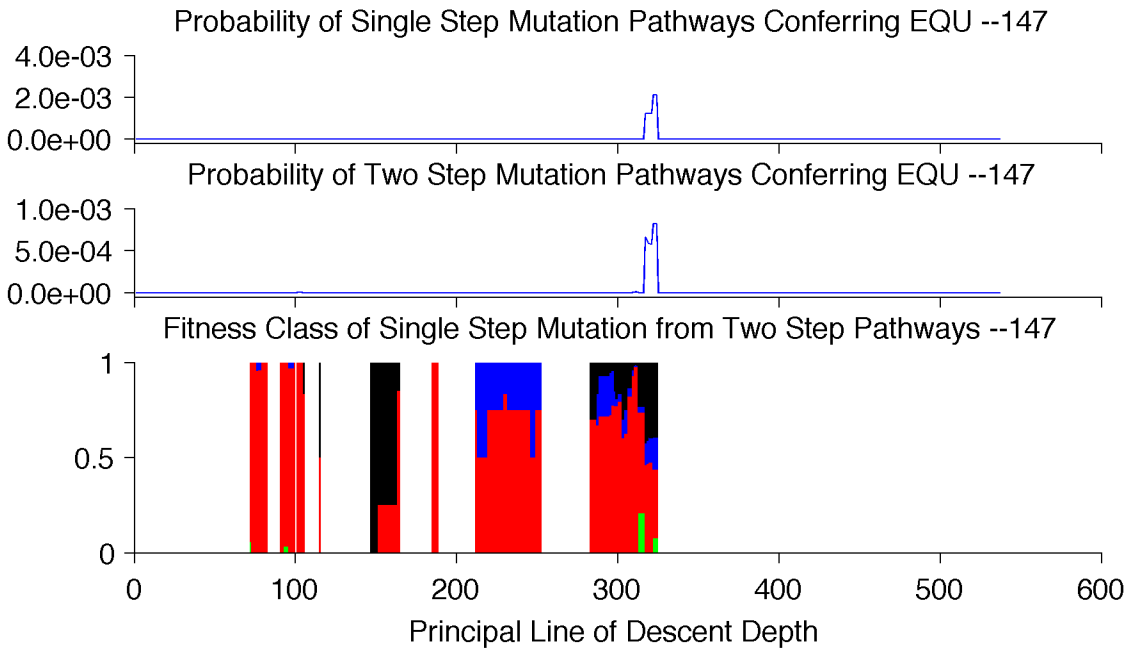

Figure 3.4: Mutational Neighborhood Group II Example — Population 147
Fitness classes: beneficial (green), neutral (blue), deleterious (red), lethal (black).

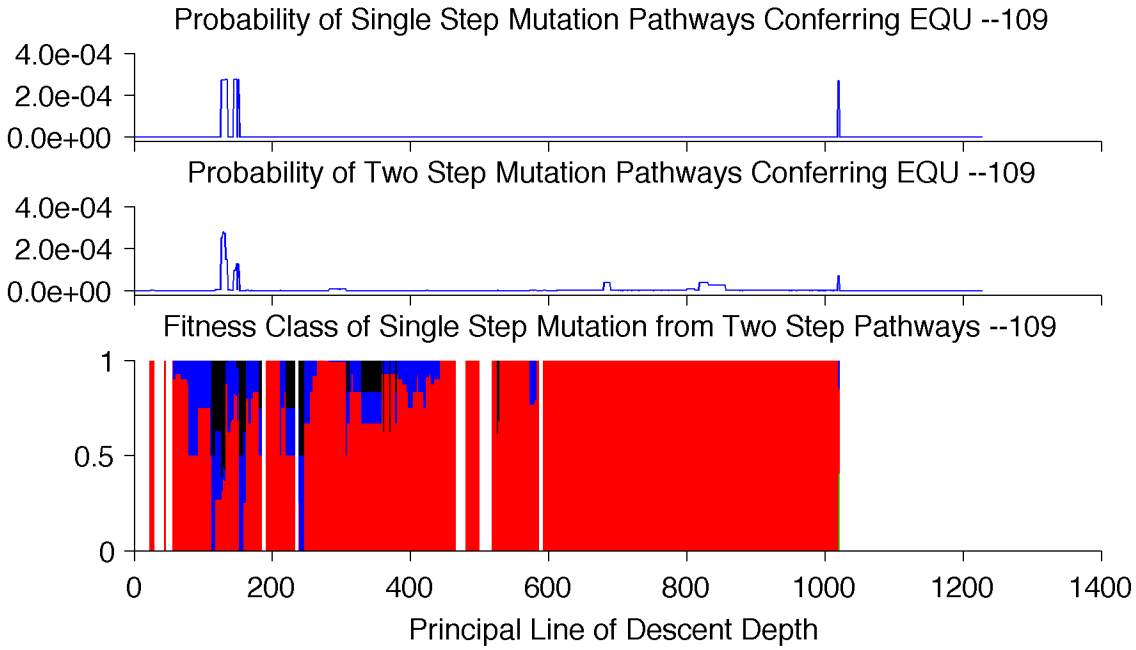

Figure 3.5: Mutational Neighborhood Group III Example — Population 109
Fitness classes: beneficial (green), neutral (blue), deleterious (red), lethal (black).

3.5), although the character of the pathways prior to the evolution of the task were similar in probability and mutation class fractions to the other five. Interestingly, population 109 was within a single mutation of EQU early in the lineage, although this mutation itself would have been highly deleterious, despite the fact that it would have conferred EQU.

Group IV, containing four lineages, demonstrated scattered regions in the histories where the genotypes were within reach of the EQU task. The pathways identified were all either deleterious or lethal when taken a single mutation at a time. None of the four populations evolved EQU during the original experiment. Figure 3.6 shows a representative line of descent.

The three lineages categorized as Group V showed small pockets of genotypes early on that were within two mutations of EQU, followed by many hundreds of genotypes represent-

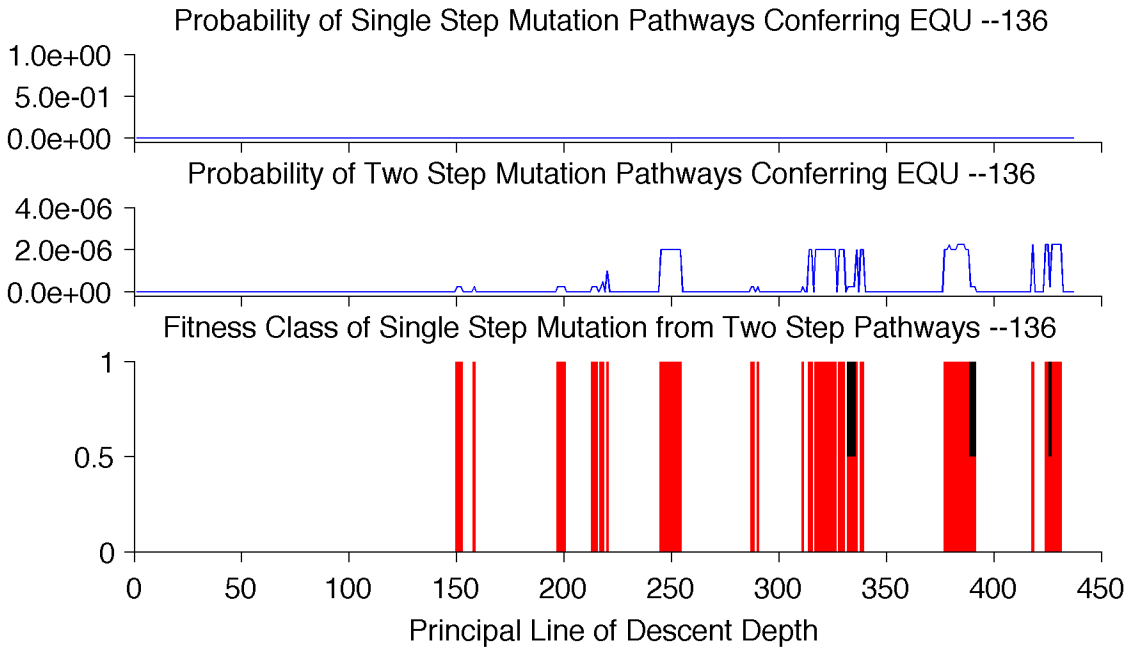

Figure 3.6: Mutational Neighborhood Group IV Example — Population 136
Fitness classes: beneficial (green), neutral (blue), deleterious (red), lethal (black).

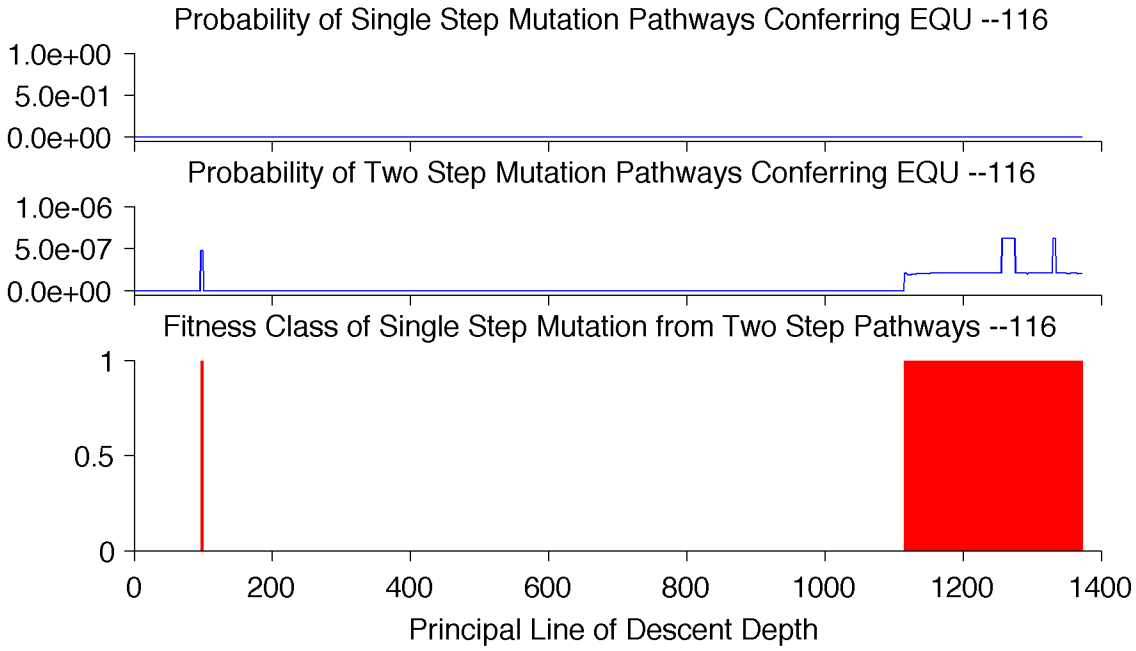

Figure 3.7: Mutational Neighborhood Group V Example — Population 116
Fitness classes: beneficial (green), neutral (blue), deleterious (red), lethal (black).

ing hundreds of thousands of updates of evolutionary time prior to the genotypes returning to a region of the landscape near the task toward the end of the experiment. The mutation pathways were almost universally deleterious. Although these populations were within two mutations of EQU at the end, often for hundreds of steps, none of them evolved it during the initial experiment. Figure 3.7 shows a representative lineage from this group.

Group VI represents two lines of descent where a segment of the genotypes early on in the experiment were within two steps of EQU. These lineages, however, wandered away from this region and were never within two mutations again. The pathways in population 113 lineage were all deleterious in nature, the other, population 125, demonstrating a small fraction of neutral pathways for some of the genotypes (Figure 3.8).

Four lines of descent fit into Group VII, demonstrating long contiguous regions of geno-

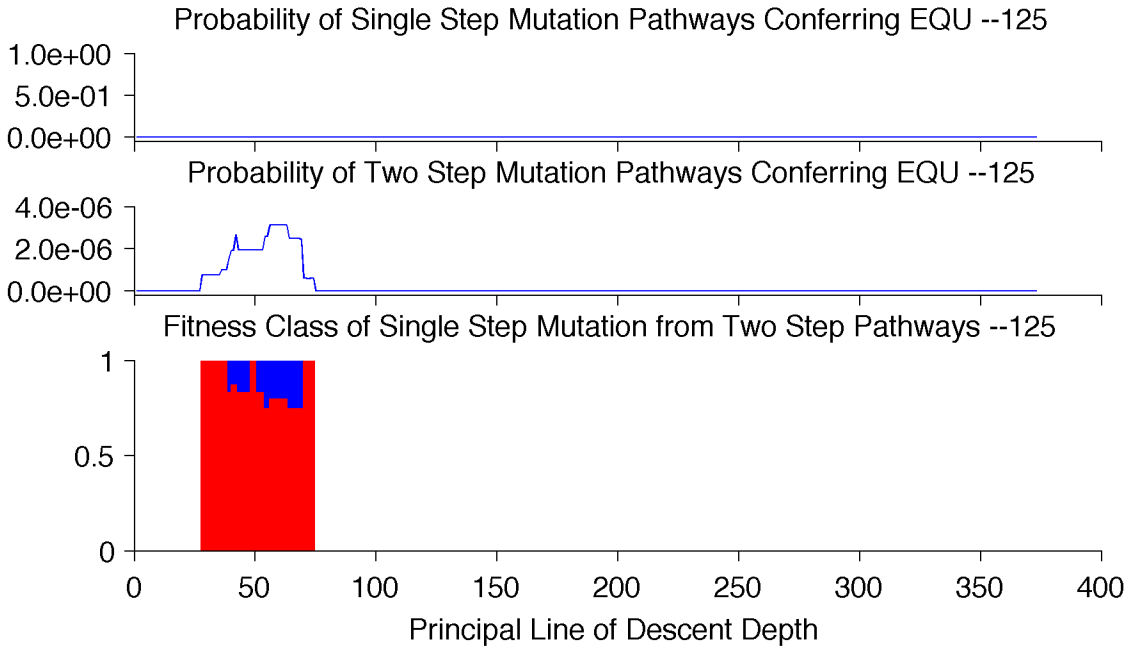

Figure 3.8: Mutational Neighborhood Group VI Example — Population 125
Fitness classes: beneficial (green), neutral (blue), deleterious (red), lethal (black).

types that are within two mutations of EQU. These four lineages can be subdivided into two groups, based on the character of the pathways within the regions. Two of the lineages had only deleterious and lethal paths identified, neither of which evolved EQU in the original experiment, despite the long stretch of time represented in the lineage. The other two lines of descent had pathways primarily consisting of deleterious mutations, accompanied by small fractions of the other mutation classes. Both also presented with single-step pathways for most of the genotypes as well, and both eventually evolved EQU at the end of the long stretch of lineage. Figure 3.9 shows a representative from the latter subgroup.

Finally Group VIII consisted of 14 lines of descent that were rarely or never within two mutations of EQU. In fact, only two of the lineages demonstrated any such genotypes. One lineage contained a single genotype with two-step pathways to EQU, all of which were

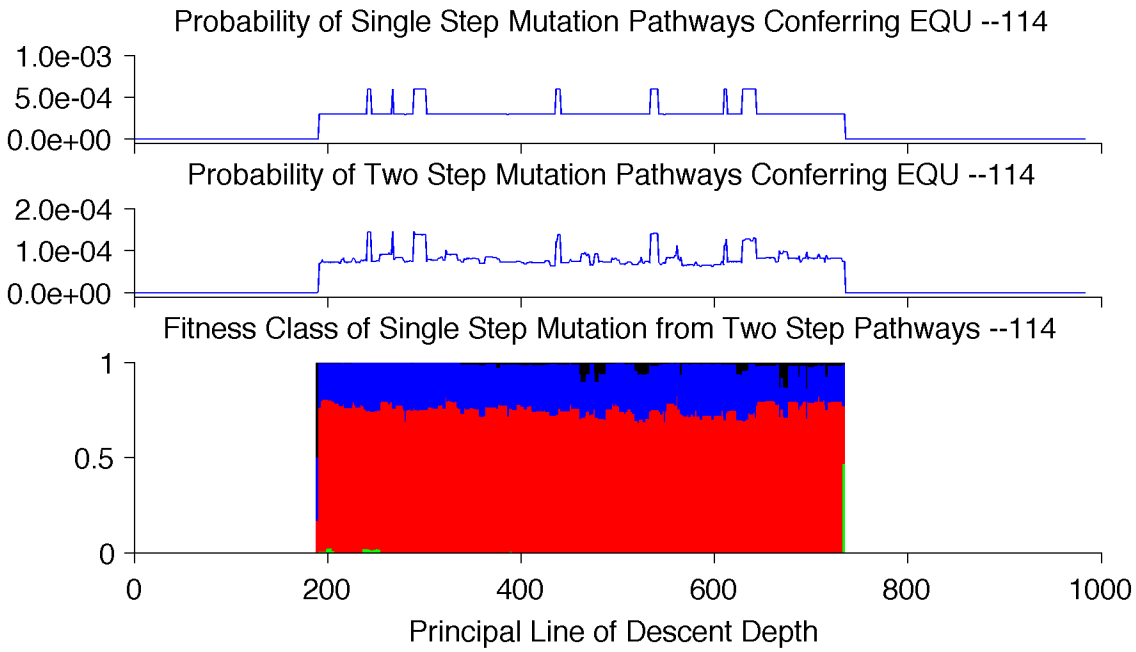

Figure 3.9: Mutational Neighborhood Group VII Example — Population 114
Fitness classes: beneficial (green), neutral (blue), deleterious (red), lethal (black).

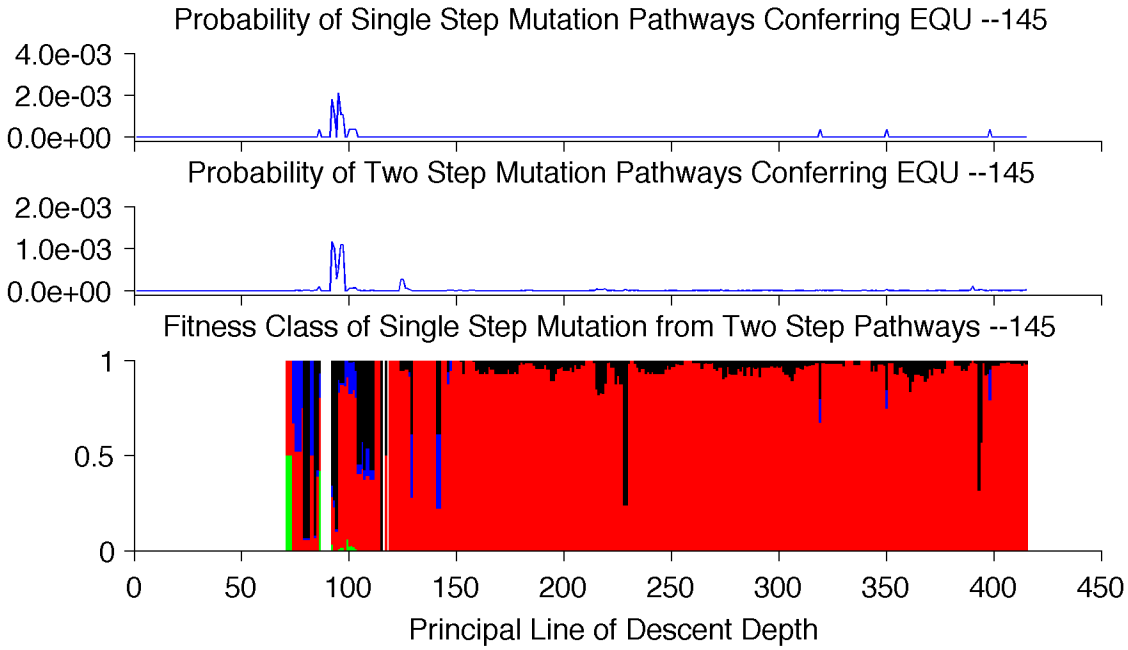

Figure 3.10: Mutational Neighborhood of the PLoD from Population 145
Fitness classes: beneficial (green), neutral (blue), deleterious (red), lethal (black).

deleterious. The other lineage also contained only a single genotype with two-step pathways, but in this case the pathways were all lethal.

One population from the first group, population 145, presents an interesting case study (Figure 3.10). It initially appeared as if it would be in Group I, with buildup of EQU likelihood, culminating the evolution of the function at around genotype 80 or so in the lineage. Extended mutational neighborhood analysis following the appearance of the EQU function showed that after about half a dozen genotypes, the EQU function was traded off for a group of three other functions, AndNot, NOR, and XOR. The genotypes in the lineage stayed within two mutations of EQU for the remainder of the experiment, but it was never re-acquired, despite occasional neutral paths.

3.2.3 STEP Sampling

I performed STEP sampling utilizing the same 50 principal lines of decent as used for the mutational neighborhood analysis, with 10 replicate STEP sampling runs based on each genotype along all 50 principal lines. For example, the 2,708 genotypes on the principal line of descent of population 113 were sampled with a total of 27,080 sample runs. I extracted the most abundant genotype from the population at the end of each STEP sample run and evaluated it in a test environment for the ability to perform the EQU function. I classified the results from these analyses into three general groups (Table 3.2).

Group A, which contained 16 lines of descent, demonstrated peaks of successful sample runs early in the history, followed by long periods of no activity prior to returning to higher probabilities just before the evolution of EQU in the main run being tested. Some of the rises toward the end were only a few steps before reaching the point where 10 out of 10 sample runs

Group	# Pops	Found EQU?	Characteristic
A	16	Yes	Minimal activity in the middle of lineage
B	13	Some	Steady, low probability throughout
C	21	No	Early chances, none observed later

Table 3.2: General groups observed during STEP sampling
Each row shows the group letter, number of populations that fit into the group, and the defining characteristic of the group.

were evolving EQU. Others showed many more steps on the lineage, with gradually increasing counts of successful sample runs, before EQU was actually observed in the original lineage. Figure 3.11 shows a representative lineage.

Thirteen lineages fit into Group B, showing a moderate propensity for EQU evolution during the sample runs from genotypes throughout the history of the run. Some of the genotypes in the middle of these lineages had fairly successful sample runs, with 6 or more of the 10 runs reaching EQU. Despite these successful genotypes, only five of the thirteen eventually evolved EQU in the original lineage. Figure 3.12 shows a line of descent demonstrating the pattern.

Finally, Group C consisted of 21 lineages demonstrate a few successful sample runs early in their histories, followed by essentially zero probability in the middle and late periods. Many of the lineages had only one or two successful sample runs, with one lineage containing none. Figure 3.13 shows one of the most successful lines of descent from this group.

The complete results for all 50 replicates are presented in Appendix B.

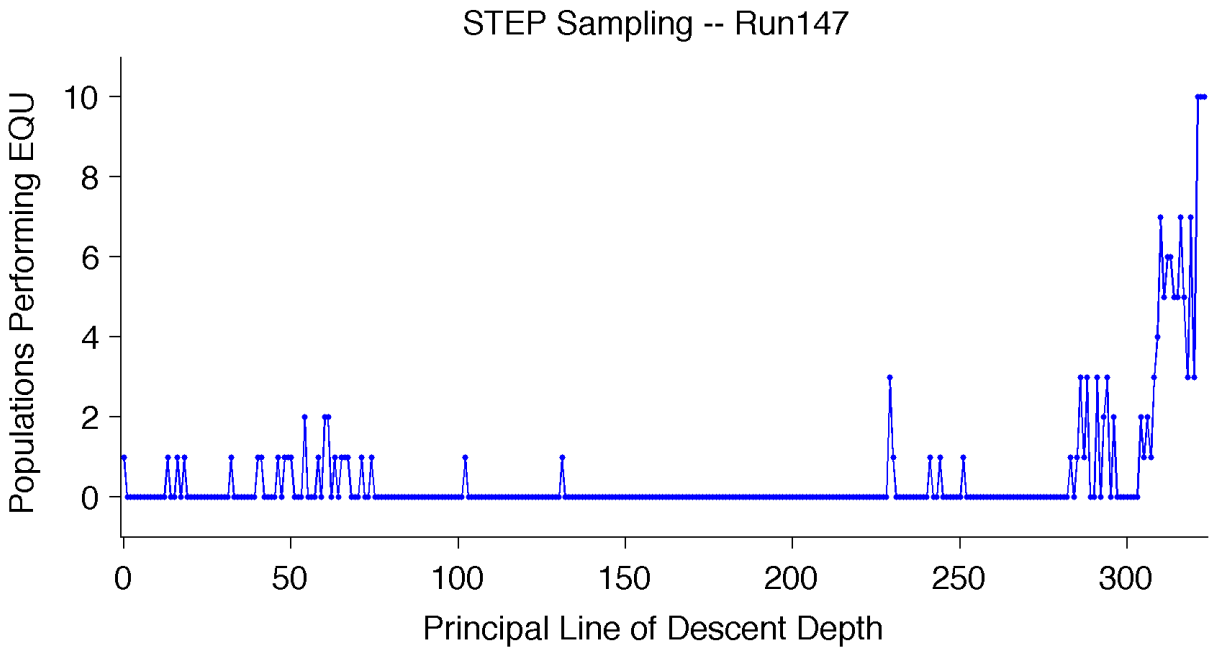

Figure 3.11: STEP Sampling Group A Example
STEP sampling along the principal line of descent from population 147, representative of Group A.

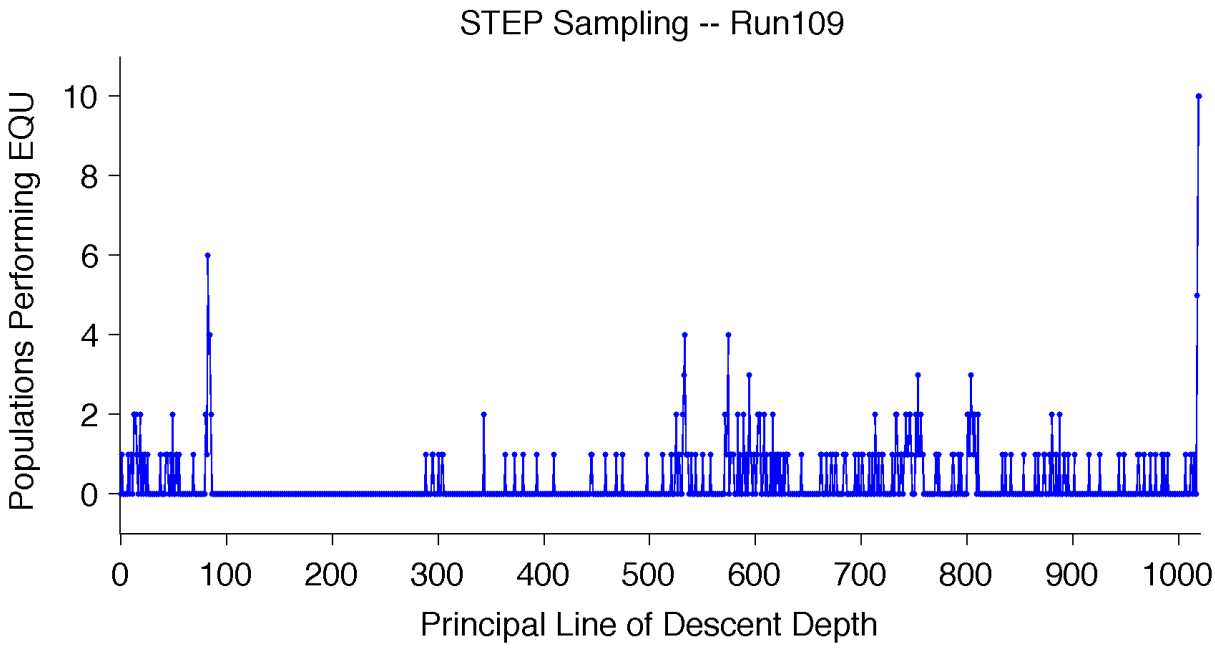

Figure 3.12: STEP Sampling Group B Example
STEP sampling along the principal line of descent from population 109, representative of Group B.

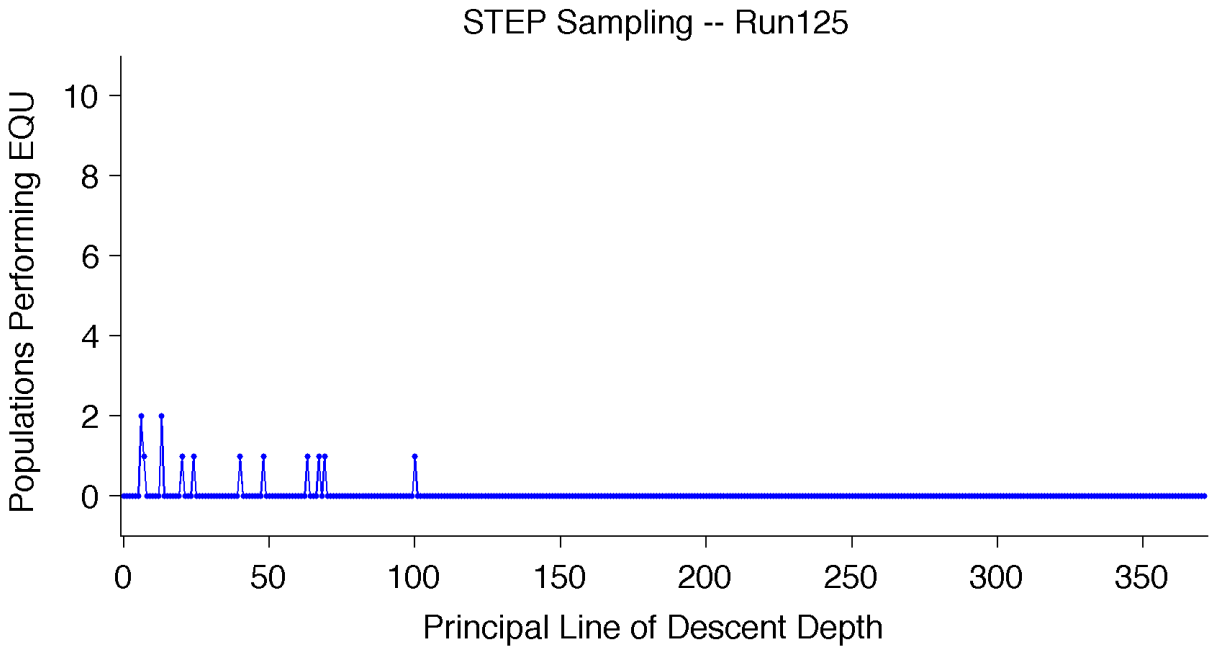

Figure 3.13: STEP Sampling Group C Example
STEP sampling along the principal line of descent from population 125, representative of Group C.

3.2.4 Discussion

In general, we found a logical consistency between our analysis of the local mutational neighborhood and STEP sampling analyses. The STEP sampling Group A, demonstrating increasingly successful samples leading up to EQU, overlaps completely with the mutational neighborhood groups with increasing pathways before EQU appeared. The other sampling and mutational neighborhood groups match up similarly. Unlike the sampling runs, the mutational neighborhood was able to identify highly unlikely pathways that were none-the-less possible. The 10 replicates used in the genotype sampling runs are likely not enough to identify such improbable events.

One mutational neighborhood group did stand out — the long window of EQU pathways along the lineage. The populations it contains came from the full spectrum of lineage samples.

Mutation Corridor		Fitness Effect				p
		Beneficial	Neutral	Deleterious	Lethal	
2 Steps to EQU	Enter	71	125	25	0	4.104×10^{-37}
2 Steps to EQU	Leave	43	107	63	0	8.049×10^{-48}
1 Step to EQU	Enter	7	22	11	0	1.530×10^{-8}
1 Step to EQU	Leave	18	14	8	0	6.398×10^{-10}
All Mutations on PLoD		3642	52714	2676	0	

Table 3.3: Mutation Corridor Entry and Exit Fitness Effects
Fitness effect of mutations identified as entries and exits to corridors in the fitness landscape near the EQU function. The last line shows the fitness effect of all mutations found on all 50 principal lines of descent examined. p-values calculated using Fisher's Exact.

These lines of descent likely demonstrate varying degrees of epistatic interactions between parts of the genome. There are pathways to EQU present for a great deal of the history sampled, yet it is still rare. Case study population 145 was initially classified into the first mutational neighborhood group, due to the early evolution of EQU. Including its full lineage, however, places it into the group of populations with long windows of genotypes with pathways to EQU in their history. The epistatic tradeoff is fairly explicit in the lineage of population 145, staying within 2 mutations of EQU for much of evolutionary time, but never able to break up the intertwined functions that killed EQU initially.

3.3 Characteristics of Potential Stepping-Stone Mutations

The mutational neighborhood analysis performed for Section 3.2.2 identified all mutations on the principal lines of decent that brought genotypes within one or two mutations of the

EQU function. All such mutations have the potential to serve as stepping stones to the evolution of EQU. I collected all mutations that served as entries to corridors within one and two mutational steps of the EQU function from all 50 lines of descent. I also identified all mutations that moved the line of descent away from regions of the fitness landscape where EQU was accessible. I assessed the fitness effect of each mutation, the results of which are presented in Table 3.3.

All four mutation categories, entering and exiting corridors within one and two mutation steps from the EQU function, demonstrated substantially over represented beneficial and deleterious mutations when compared with all mutations on the lines of descent (Fisher's Exact, see Table 3.3). For example, beneficial mutations were 5.2 times and deleterious mutations 2.4 times more common in entries to 2-step corridors relative the base distribution of mutations on the line of descent. Such beneficial mutations cannot be due to the evolution of EQU at the entry of these corridors. These results imply that mutations actively changing the quality of the genotype are more important for exploiting evolutionary potential than neutral mutations.

Chapter 4

How Does the Genetic Hardware Affect Evolutionary Potential?

The genetic hardware of an evolving system forms the basis of heritability, encompassing an encoding of the complete description of entities (a genome) and the mechanics of how that description may be translated into the entities themselves (the virtual hardware). The properties of the genetic hardware affect the robustness, flexibility, and potential complexity of those entities within the system. These factors must inherently have broad impacts on evolutionary potential, but the nature of that impact is not well understood. In natural evolution, the genetic hardware is the genome (such as DNA and RNA), the proteins that must be present for it to be translated, and the biochemistry that governs that translation. The mechanics of constructing and copying these structures play critical roles in determining properties such as the robustness of genetic material and the potential complexity of biological processes.

In evolutionary computation, the combination of how individual solutions are encoded

and the rules that are used to evaluate the fitness of those encodings define the genetic hardware. The genetic hardware may be as simple as bit strings with pre-defined meanings associated with each bit, as is commonly used in some genetic algorithms, or it may be complete programs with instruction interactions and data-flow semantics, as in genetic programming or digital evolution. All aspects of these systems are designed by engineers to provide specific capabilities deemed sufficient for solving the particular problem under consideration. These design choices may have significant impact on the ability of a system to evolve the desired outcome.

In designing evolutionary computation systems, it is critical to understand how sensitive the evolutionary process is to changes in the mechanics of the genetic hardware. Will seemingly small changes yield large differences in performance? Are certain aspects of a genetic hardware more important than others? How sensitive is genetic hardware to the environment? Is it possible for a genetic hardware to perform well generally, or is an effectively designed one critically tied to aspects of the environment? What characteristics of the genetic hardware affect the potential complexity of evolved entities? What aspects of genetic hardware promote robustness, and how do they interact with complexity? Are there general design principles that can be gleaned to help steer the design of future evolutionary computation systems?

Understanding how genetic hardware affects evolutionary potential will provide insight into the answers to these and other questions. Although the no free lunch theorem states that no "perfect" encoding exists, I expect that, due to the many arbitrary choices in its design, the evolutionary potential of the current genetic hardware in Avida is far from optimal for evolving populations in the types of environments interested to researchers and

engineers (Wolpert and Macready, 1995, 1997).

4.1 Optimizing Evolutionary Potential with Instruction Set Architectures

The design of the instruction set within a digital evolution system forms the basis of the genetic hardware upon which organisms evolve. It plays an important role in the robustness and flexibility of evolved solutions (Ofria *et al.*, 2002). As the scope of problems explored within digital evolution systems such as Avida, continues to expand, it is desirable to expand the evolutionary potential afforded by the system throughout a broad range of computational environments. Starting from the current default, the HEADS instruction set, I have systematically investigated a series of modifications to the instruction set architecture of Avida virtual CPUs, incorporating the most promising modifications into the system as I go. I have evaluated the evolutionary potential of these modified instruction sets in seven computational environments, described in Section 2.2, representing a wide range of desired capabilities by comparing the final results of experiments performed in each environment with each instruction set modification.

4.1.1 Tested Architecture Modifications

In the default HEADS instruction set, most instructions can have one aspect of their function modified by a single nop instruction that follows in the genome. I aimed to improve the flexibility by which data could be accessed and modified in the virtual CPUs by implementing the FULLY-ASSOCIATIVE (FA) instruction set. I extended the nop modification system

used by instructions so that most instructions could be modified by more than one nop. The default behavior of all instructions remains the same when followed by zero or one nop instruction. However, for arithmetic, logic, and conditional instructions that operate on more than one operand or have both source and destination registers, the FA instruction set uses additional nops, if present, to further specify these parameters. For example, an `add` instruction, by default will perform $regB = regB + regC$. If it is followed by one `nop-A`, this will alter the register where the result is placed so that it instead performs $regA = regA + regB$. When followed by `nop-A nop-C`, the add instruction would have the same behavior in the default instruction set (ignoring the `nop-C`), but in the FULLY-ASSOCIATIVE set it would instead perform $regA = regC + regA$. In this manner, very specific operations may be created, while retaining robust default behavior.

The R series of instruction set architectures build upon the FULLY-ASSOCIATIVE architecture to increase the working register set beyond the three default registers, exposing one or more additional architectural registers, up to a total of 16 in R16. The original design choice was made to minimize the number of registers in order to simplify the complexity of using them, but a larger number of registers has not previously been systematically tested. For each additional register, I added a corresponding nop instruction to the instruction set (`nop-D`, `nop-E`, `nop-F`, `nop-G`, etc.). None of the default registers used by the instruction set were altered, meaning that these additional registers can be accessed only when the new nop instructions are used to modify an instruction. Since nop modification is also used for head selection, the additional `nop-D` in the R4 architecture provides direct access to the FLOW head. In the R5 through R16 architectures, extra unassigned heads that may be used as genome place-markers are available for each additional nop instruction.

The LABEL series of instruction set architectures extends the R6 architecture (which proved to be the most effective, as described in the results below), explicitly separating genome labels from nop sequences used to modify instruction operands. The goal of this change was to prevent labeled genome positions, especially those used for self-replication, from otherwise conflicting with instruction argumentation as facilitated by the FULLY-ASSOCIATIVE architecture. Instructions that operate on genome labels, `search-seq-comp-s` and `if-copied-seq-comp`, were extended with variants (`search-lbl-comp-s` and `if-copied-lbl-comp`) that recognize sequences of nop instructions only if they begin with the special `label` instruction. When executed directly, the `label` instruction performs no operation. The LABEL-DIRECT series architectures alter the pattern matching algorithm from the default of cyclic-complementary to direct sequence matching. For example, LABEL-SEQ-DIRECT replaces `search-seq-comp-s` and `if-copied-seq-comp` with `search-seq-direct-s` and `if-copied-seq-direct`, respectively, while retaining `search-lbl-comp-s` and `if-copied-lbl-comp`. The LABEL-BOTH architectures include both pattern matching algorithm instruction variants. In order to increase the power of labeled execution flow, all LABEL series instruction sets omit the `set-flow` instruction that performs absolute addressing.

The SPLIT-IO instruction set architecture alters the LABEL-SEQ-DIRECT architecture, splitting the `IO` instruction into two separate `input` and `output` instructions. Both of the new instructions use the same default register location as the `IO` instruction and can each be modified by one nop.

The SEARCH series of instruction set architectures extend the SPLIT-IO architecture with enhanced searching and jumping capabilities. The SEARCH-DIRECTIONAL set adds

Instruction Set	If0	IfX	MovHead
Flow-If0	•		
Flow-IfX		•	
Flow-MvH			•
Flow-If0-MvH	•	•	
Flow-IfX-MvH		•	•
Flow-If0-IfX-MvH	•	•	•

Table 4.1: Flow series instruction sets tested
Instruction set by row, with marks in each column indicating that the set contains the relevant instruction group.

two pairs of directional `search-` instructions that scan the genome forward or backward relative to the instruction pointer for a label or sequence match. The Search-Goto set, adds a single `goto` instruction that reads the nop sequence that follows the instruction, if present, and will unconditionally jump to the first genome location following the matching label that begins with a `label` instruction. If no matching label is found, execution ignores the goto instruction. The Search-GotoIf group adds two conditional `goto` variants, `goto-if-n-equ` and `goto-if-less`, that execute the jump only if the conditional test evaluates to *true*.

The Flow series of instruction set architectures builds upon the flow control features of the `Search-Directional` architecture, testing multiple combinations of additional flow control instructions (Table 4.1). The If0 group adds four single argument `if` instructions, `if-not-0`, `if-equ-0`, `if-gtr-0`, and `if-less-0`, that conditionally execute the following instruction based on the comparison of the argument with 0. The IfX group adds two if variants, `if-gtr-x` and `if-equ-x`, that conditionally execute the following instruction based on the result of comparing *regB* with a nop modified number. The default value used by `if-gtr-x` and `if-equ-x` is a 1. For each nop in the label following a given `if-gtr-x` or `if-equ-x` instruction, the bit is left shifted 1, 2, or 3 times for each `nop-B`, `nop-C`, or

`nop-D`, respectively. Whenever a `nop-A` is found in the label sequence, the sign-bit of the value is toggled. Finally, the MovHead group adds two conditional `mov-head` variants, `mov-head-if-n-equ` and `mov-head-if-less`, that operate similarly to the conditional `goto` instructions.

4.1.2 Methods

I tested each instruction set architecture with 200 replicate populations in each of the seven computational environments, Logic-9, Logic-77, Match-12, Fibonacci-32, Sort-10, Limited-9, and Navigation. The populations evolved on 60x60 worlds for 100,000 updates. Each population was seeded with a single ancestral organism capable only of self-replication. Small variations in the initial genotype used in each architecture were often necessary, due to the functional differences among the instruction sets, but I made a strong effort to limit these variations specifically to neutral labeling instructions used in self-replication.

4.1.3 Results

I have focused on two measures to evaluate how well populations solved the computational challenges of the environment when evolved with each instruction set architecture: mean fitness and task success. Both measure ability of the evolved organisms to perform tasks within the environment.

Mean fitness averages the fitness values of each living organism in the population at the moment the experiment finished. It takes into account both the computational capability of the organism and the efficiency of self-replication, also called the "gestation time". We examined the distributions of these fitness values for all instruction set variants in each

environment. For each modified instruction set, we compared the 200 population fitness values with those of a reference instruction set architecture using a Wilcoxon rank-sum test. We determined significance using $a = 0.05$ with sequential Bonferroni correction. Confidence intervals, as shown in tables below, represent 2.5% and 97.5% quantiles that we generated using non-parametric bootstrap with 10,000 iterations.

Task success is a direct examination of the computational capabilities of the organisms within the final population, for the specific environment of the experiment. We measure the task success of a population as the sum of the qualities by which the average organism performs each task. To calculate a task success t_p of population p, we determine each organism's quality at each task and then sum over these values, finally dividing by the total number of organisms in the population. More formally,

$$t_p = \sum_{i=1}^{N_p} \sum_{j=1}^{T} \frac{q_{i,j}}{N_p} \tag{4.1}$$

where N_p is the number of organisms in population p, T is the number of tasks in the environment, and $q_{i,j}$ is the quality q at which organism i is performing task j. Task quality (q) is a value between 0 and 1, where 1 means the organism has found a perfect solution for a task. Environments that support near-matches use task quality to adjust the metabolic reward accordingly. The maximum task success for a given environment is equal to the total number of tasks rewarded in that environment; for example the maximum task success of the Logic-9 environment is 9. Normalized task success, as presented in the following results, divides the observed task success by the maximum in each environment. Similar to population mean fitness, we compared the distribution of task success of each instruction set to the control architecture using a Wilcoxon rank-sum test, sequential Bonferroni correction,

and non-parametric bootstrap confidence intervals.

4.1.3.1 Fully-Associative Argumentation

In conducting my analysis, the FULLY-ASSOCIATIVE (FA) instruction set, which addresses the flexibility of register data flow, shows significant improvement in six of the seven environments (Tables 4.2 and 4.3). The logic-based environments, Logic-9, Logic-77, and Limited-9, all show substantially improved fitness and task success. The Logic-77 environment in particular, benefits from the FA instruction set with nearly 2.9 times increase in median task success and dramatically increased average fitness. The fully-associative capability, facilitating specific instruction formats, appears crucial within the highly diverse Logic-77 environment. Indeed, on average 9.2% of the nop-modified instructions in the dominant genotype at the end of the FA experiments with the Logic-77 environment used more than one nop modifier. The Fibonacci-32 environment also sees a notable 44% improvement in task success, with a corresponding increase in fitness. Mean usage of multiple nop modifiers was 16.4% in the final dominant genotype of the Fibonacci runs. The Sort-10 and Match-12 environments show statistically significant gains for both metrics, but none of these improvements are substantial in nature. The Navigation environment shows a slight, non-significant decline in fitness ($p < 0.054$, Wilcoxon rank-sum test) and task success ($p < 0.178$, Wilcoxon rank-sum test) when tested with the FA instruction set.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
HEADS	19.07 (17.71, 19.76)	12.43 (11.51, 14.22)	0.173 (0.146, 0.224)	3.730 (3.300, 4.050)	-0.54 (-0.63, -0.45)	4.430 (4.283, 4.595)	1.071 (1.035, 1.383)
FA	**22.99** (22.70, 23.08)	**39.35** (35.05, 41.83)	**0.215** (0.919, 0.251)	**4.806** (4.474, 5.212)	**-0.38** (-0.45, -0.33)	**4.840** (4.671, 5.082)	1.038 (1.022, 1.069)

Table 4.2: HEADS and FULLY-ASSOCIATIVE Architectures Fitness

Fitness results of the HEADS and FULLY-ASSOCIATIVE (FA) instruction set architectures, where multiple nop arguments can modify the behavior of an instruction. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations after sequential Bonferroni correction.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
HEADS	0.829 (0.752, 0.839)	0.176 (0.161, 0.198)	0.145 (0.145, 0.146)	0.206 (0.178, 0.238)	1.31×10^{-4} (1.08, 1.47)	0.909 (0.894, 0.913)	3.97×10^{-3} (3.96, 4.35)
FA	**0.936** (0.930, 0.943)	**0.505** (0.453, 0.546)	**0.148** (0.147, 0.149)	**0.297** (0.278, 0.332)	1.55×10^{-4} (1.44, 1.67)	**0.927** (0.924, 0.929)	3.96×10^{-3} (3.95, 3.97)

Table 4.3: HEADS and FULLY-ASSOCIATIVE Architectures Task Success

Task success results of the HEADS and FULLY-ASSOCIATIVE (FA) instruction set architectures. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries denote significant ($p < 0.05$, Wilcoxon rank-sum test) deviations.

4.1.3.2 Number of Registers

The R series instruction sets generally show little variation in performance (Tables 4.4 and 4.5). In the Logic-77 environment there is a slight positive trend as the number of registers increases, but none are significant after Bonferroni correction, and the magnitudes of the changes are not particularly notable. The only substantial differences observed among all tested configurations are a drop in task success and a drop in fitness with R16 in the Logic-9 environment, indicating a potential drag on the system due to the dramatic increase in instruction set size with the addition of 13 more nops. The Sort-10 environment demonstrates significant loss of performance in all treatments, relative to the FA architecture, though none of the variation observed is substantial in nature ($\ll 1\%$ difference in task success). The Navigation environment does show what initially appears to be a substantial uptick in performance under R16, but with task success still well below 1%, it is not enough to allow the populations to complete this task. It does, however, indicate that we may wish to explore higher register counts again in configurations where populations have more success with this task.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
FA	22.63 (22.39, 23.01)	38.85 (34.76, 43.67)	0.223 (0.191, 0.273)	4.828 (4.255, 5.332)	0.09 (-0.07, 0.21)	4.934 (4.661, 5.282)	1.027 (1.016, 1.042)
R4	22.85 (22.67, 23.01)	38.70 (33.90, 43.02)	0.243 (0.204, 0.290)	4.666 (4.233, 5.142)	**-0.39** (-0.48, -0.32)	5.253 (4.925, 5.514)	**1.080** (1.054, 1.792)
R5	22.73 (22.50, 22.86)	38.42 (34.43, 42.54)	0.231 (0.206, 0.281)	5.067 (4.540, 5.623)	**-0.45** (-0.57, -0.39)	5.158 (4.300, 5.400)	**1.083** (1.056, 1.340)
R6	22.78 (22.29, 22.97)	43.01 (40.01, 45.90)	0.229 (0.206, 0.274)	4.908 (4.293, 5.719)	**-0.43** (-0.54, -0.34)	5.117 (4.925, 5.374)	**1.111** (1.080, 2.730)
R7	22.75 (22.58, 22.97)	43.41 (38.97, 45.67)	0.204 (0.177, 0.225)	4.598 (4.174, 5.078)	**-0.40** (-0.49, -0.29)	5.135 (4.978, 5.407)	**1.562** (1.096, 3.234)
R8	22.75 (22.55, 22.95)	43.04 (39.25, 47.78)	**-0.027** (-0.07, 0.19)	4.831 (4.392, 5.308)	**-0.47** (-0.57, -0.33)	5.292 (5.058, 5.736)	**1.156** (1.099, 2.815)
R12	22.62 (22.45, 22.76)	44.26 (40.82, 48.18)	**-0.11** (-0.12, -0.08)	4.678 (4.082, 5.244)	**-0.54** (-0.56, -0.49)	5.180 (4.901, 5.621)	**1.377** (1.114, 3.012)
R16	**21.68** (19.76, 22.22)	42.26 (40.02, 46.26)	**-0.11** (-0.13, -0.10)	4.028 (3.620, 4.474)	**-0.55** (-0.59, -0.50)	**5.734** (5.390, 6.466)	**3.78** (1.157, 3.326)

Table 4.4: R Series Architectures Fitness

Fitness results of the R series instruction set architectures, which vary the number of registers available in the virtual CPUs. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations after sequential Bonferroni correction.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
FA	0.932 (0.921, 0.938)	0.495 (0.452, 0.565)	0.147 (0.146, 0.148)	0.288 (0.263, 0.307)	2.53×10^{-4} (2.21, 2.74)	0.926 (0.923, 0.929)	3.96×10^{-3} (3.95, 3.96)
R4	0.937 (0.929, 0.941)	0.506 (0.441, 0.554)	0.146 (0.145, 0.148)	0.276 (0.256, 0.289)	$\mathbf{1.52\times10^{-4}}$ (1.42, 1.67)	0.923 (0.920, 0.929)	$\mathbf{3.97\times10^{-3}}$ (3.96, 5.05)
R5	0.936 (0.929, 0.940)	0.493 (0.450, 0.544)	**0.145** (0.144, 0.147)	0.300 (0.284, 0.327)	$\mathbf{1.38\times10^{-4}}$ (1.09, 1.62)	0.927 (0.923, 0.929)	$\mathbf{3.97\times10^{-3}}$ (3.96, 4.20)
R6	0.932 (0.927, 0.940)	0.563 (0.521, 0.592)	**0.145** (0.144, 0.147)	0.294 (0.268, 0.326)	$\mathbf{1.49\times10^{-4}}$ (1.14, 1.160)	0.930 (0.926, 0.932)	$\mathbf{3.98\times10^{-3}}$ (3.97, 6.68)
R7	0.940 (0.930, 0.943)	0.554 (0.502, 0.592)	**0.144** (0.142, 0.146)	0.281 (0.247, 0.305)	$\mathbf{1.51\times10^{-4}}$ (1.26, 1.63)	0.928 (0.923, 0.932)	$\mathbf{4.47\times10^{-3}}$ (3.98, 7.65)
R8	0.938 (0.931, 0.942)	0.555 (0.504, 0.613)	**0.079** (0.078, 0.143)	0.299 (0.275, 0.323)	$\mathbf{1.33\times10^{-4}}$ (1.06, 1.57)	0.927 (0.924, 0.929)	$\mathbf{3.99\times10^{-3}}$ (3.97, 6.19)
R12	0.939 (0.933, 0.943)	0.575 (0.525, 0.613)	**0.078** (0.077, 0.078)	0.298 (0.268, 0.318)	$\mathbf{1.06\times10^{-4}}$ (1.03, 1.11)	0.930 (0.928, 0.933)	$\mathbf{4.31\times10^{-3}}$ (3.98, 7.33)
R16	0.910 (0.854, 0.931)	0.550 (0.524, 0.589)	**0.077** (0.077, 0.078)	0.269 (0.237, 0.302)	$\mathbf{1.05\times10^{-4}}$ (1.01, 1.08)	0.928 (0.925, 0.932)	$\mathbf{7.31\times10^{-3}}$ (3.99, 7.81)

Table 4.5: R Series Architectures Task Success

Task success results of the R series instruction set architectures. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries denote significant ($p < 0.05$, Wilcoxon rank-sum test) deviations.

4.1.3.3 Explicit Labels

The LABEL series instruction sets show mixed results (Tables 4.6 and 4.7). The Logic-9, Limited-9, Sort-10, and Navigation environments show virtually no substantial differences in task success, regardless of the set used. The Limited-9, Sort-10, and Navigation environments shows slight positive fitness trends as more labeling options are included in the instruction set. The Logic-77 environment shows significantly detrimental results for both fitness and task success when only the label-based instructions are included. When any form of sequence matching instructions are included in the Logic-77 environment, both metrics return to the reference levels. The Match-12 environment shows no significant difference for either metric among all but one instruction set. LABEL-SEQ-BOTH, the most complete instruction set in this group, shows a notably significant drop of both metrics in the Match environment. The Fibonacci-32 environment shows positive gains in all LABEL series instruction sets. The positive gains observed in the Fibonacci-32 environment were both significant and substantial, with 24.6% and 27.2% improvement in fitness and task success, respectively, when using the LABEL-SEQ-BOTH instruction set. In the Navigation environment using the LABEL-SEQ-DIRECT instruction set, 8 outlier populations notably demonstrated task success greater than 0.10, with two at 0.141, indicating that substantial progress was made in those particular runs. No previous runs in this environment have exhibited such success in the short time period of 100,000 updates used (Grabowski *et al.* (2010, 2011), plus unpublished data).

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
R6	22.59 (22.10, 22.85)	38.42 (34.96, 44.12)	0.216 (0.201, 0.257)	4.572 (3.904, 5.134)	-0.22 (-0.32, -0.09)	5.205 (5.016, 5.518)	1.537 (1.097, 3.261)
LABEL	22.67 (22.42, 22.94)	**28.70** (25.66, 33.37)	0.248 (0.218, 0.316)	**6.213** (5.669, 6.648)	-0.42 (-0.52, -0.33)	5.430 (5.195, 5.630)	1.926 (1.108, 3.313)
LABEL-DIRECT	22.50 (21.84, 22.74)	**27.20** (24.38, 31.32)	0.215 (0.189, 0.252)	**5.435** (4.545, 6.174)	**-0.46** (-0.54, -0.40)	5.784 (5.429, 6.325)	1.087 (1.064, 2.742)
LABEL-BOTH	22.32 (19.68, 22.61)	**32.26** (28.96, 38.07)	0.203 (0.163, 0.230)	**6.403** (5.715, 6.606)	**-0.47** (-0.56, -0.38)	5.553 (5.098, 5.885)	3.084 (1.148, 3.396)
LABEL-SEQ	22.43 (21.94, 22.66)	40.26 (35.12, 44.36)	0.207 (0.134, 0.257)	**6.357** (5.797, 6.733)	-0.29 (-0.36, -0.15)	5.438 (5.079, 5.755)	2.203 (1.089, 3.161)
LABEL-SEQ-DIRECT	22.46 (22.23, 22.66)	44.13 (39.73, 48.52)	0.198 (0.126, 0.217)	**6.175** (5.212, 6.531)	-0.33 (-0.44, -0.21)	5.651 (5.441, 5.967)	2.335 (1.077, 3.335)
LABEL-DIRECT-SEQ	22.53 (22.25, 22.69)	41.74 (38.58, 44.36)	0.210 (0.183, 0.300)	**5.933** (5.135, 6.495)	-0.40 (-0.47, -0.23)	5.528 (5.528, 5.968)	2.319 (1.093, 3.235)
LABEL-SEQ-BOTH	22.44 (21.75, 22.68)	39.56 (36.64, 42.72)	**-0.094** (-1.42, 0.088)	**6.116** (5.336, 6.430)	-0.33 (-0.45, -0.17)	**5.990** (5.621, 6.374)	3.173 (2.955, 3.295)

Table 4.6: LABEL Series Architectures Fitness

Fitness results of the LABEL series instruction set architectures. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations after sequential Bonferroni correction.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
R6	0.926 (0.908, 0.937)	0.505 (0.461, 0.574)	0.144 (0.142, 0.145)	0.278 (0.241, 0.303)	1.86×10^{-4} (1.62, 2.10)	0.930 (0.926, 0.932)	4.33×10^{-3} (3.97, 7.76)
LABEL	0.941 (0.934, 0.945)	**0.389** (0.352, 0.450)	0.146 (0.144, 0.147)	**0.380** (0.342, 0.396)	$\mathbf{1.45\times10^{-4}}$ (1.14, 1.57)	0.932 (0.928, 0.934)	4.58×10^{-3} (3.98, 7.68)
LABEL-DIRECT	0.937 (0.916, 0.943)	**0.366** (0.329, 0.416)	0.145 (0.144, 0.146)	**0.335** (0.295, 0.383)	$\mathbf{1.34\times10^{-4}}$ (1.10, 1.57)	0.934 (0.931, 0.936)	3.98×10^{-3} (3.97, 6.16)
LABEL-BOTH	0.922 (0.857, 0.939)	**0.422** (0.385, 0.495)	0.145 (0.140, 0.146)	**0.388** (0.367, 0.403)	$\mathbf{1.37\times10^{-4}}$ (1.09, 1.51)	0.932 (0.929, 0.935)	7.05×10^{-3} (4.00, 7.96)
LABEL-SEQ	0.932 (0.919, 0.938)	0.509 (0.460, 0.573)	0.143 (0.139, 0.144)	**0.397** (0.365, 0.405)	1.68×10^{-4} (1.57, 2.02)	0.928 (0.925, 0.929)	5.31×10^{-3} (3.98, 7.57)
LABEL-SEQ-DIRECT	0.929 (0.918, 0.939)	0.559 (0.522, 0.612)	0.144 (0.141, 0.146)	**0.370** (0.309, 0.398)	1.61×10^{-4} (1.49, 1.81)	0.931 (0.928, 0.934)	5.06×10^{-3} (3.98, 7.70)
LABEL-DIRECT-SEQ	0.932 (0.919, 0.941)	0.542 (0.500, 0.562)	0.143 (0.141, 0.145)	**0.359** (0.300, 0.399)	1.53×10^{-4} (1.36, 1.77)	0.930 (0.928, 0.933)	5.24×10^{-3} (3.98, 7.63)
LABEL-SEQ-BOTH	0.926 (0.914, 0.934)	0.517 (0.482, 0.545)	**0.079** (0.078, 0.125)	**0.374** (0.300, 0.398)	1.64×10^{-4} (1.51, 1.94)	0.928 (0.923, 0.930)	7.74×10^{-3} (7.20, 7.91)

Table 4.7: LABEL Series Architectures Task Success

Task success results of the LABEL series instruction set architectures. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries denote significant ($p < 0.05$, Wilcoxon rank-sum test) deviations.

4.1.3.4 Split Input/Output Operations

The SPLIT-IO instruction set shows improvements that are both significant and often substantial in the Logic-9 and Logic-77 environments, the Match-12 environment, and the Fibonacci-32 environment (Tables 4.8 and 4.9). Indeed the Logic-77 and Fibonacci-32 environments show 21% and 17% improvements in median task success, respectively. The Sort-10 environment, on the other hand, completely collapses, showing effectively 0 task success and correspondingly low fitness. The Limited-9 environment shows mixed results, with a small gain in task success but a drop in fitness. The Navigation environment shows marginal drops in both metrics, though neither significant and, similar to previous instruction sets tested, still well below 1% of the success possible.

4.1.3.5 Search

The three SEARCH series instruction sets showed little measurable difference in performance for the Logic-9, Match-12, Fibonacci-32, Sort-10, Limited-9, and Navigation environments (Tables 4.10 and 4.11). The Logic-77 environment showed small, significant drops in fitness for all sets, with a corresponding drop in task success.

In the SEARCH-GOTO instruction set, I initially tested a variant of the `jmphead` instruction, which changed the default head it operated on to be the FLOW head. A notable and often significant drop in fitness was observed in all seven environments with these two instruction sets, leading to the architectures explored here.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
Label-Seq-Direct	22.56 (22.32, 22.72)	43.57 (39.73, 46.60)	0.207 (0.182, 0.239)	6.106 (5.326, 6.549)	-0.32 (-0.39, -0.24)	5.812 (5.390, 6.169)	2.641 (1.198, 3.341)
Split-IO	**23.07** (22.87, 23.22)	**53.94** (50.34, 56.72)	**0.337** (0.314, 0.360)	**8.096** (7.983, 8.207)	**-1.03** (-1.03, -1.02)	5.343 (5.221, 5.520)	1.091 (1.062, 2.920)

Table 4.8: Split-IO Architecture Fitness

Fitness results of the Label-Seq-Direct and Split-IO instruction set architectures. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations after sequential Bonferroni correction.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
Label-Seq-Direct	0.930 (0.920, 0.935)	0.559 (0.521, 0.593)	0.145 (0.142, 0.146)	0.384 (0.318, 0.397)	1.62×10^{-4} (1.52, 1.74)	0.926 (0.922, 0.928)	6.63×10^{-3} (3.99, 7.94)
Split-IO	**0.940** (0.936, 0.942)	**0.678** (0.651, 0.707)	**0.148** (0.148, 0.149)	**0.449** (0.447, 0.461)	**0.0** (0.0, 0.0)	**0.931** (0.927, 0.933)	3.99×10^{-3} (3.97, 7.41)

Table 4.9: Split-IO Architecture Task Success

Task success results of the Label-Seq-Direct and Split-IO instruction set architectures. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries denote significant ($p < 0.05$, Wilcoxon rank-sum test) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
SPLIT-IO	23.19 (23.05, 23.25)	54.58 (52.46, 58.54)	0.307 (0.261, 0.335)	8.139 (8.027, 8.318)	-1.03 (-1.04, -1.02)	5.477 (5.014, 5.912)	2.909 (1.121, 3.382)
SEARCH-DIRECTIONAL	23.02 (22.87, 23.17)	**48.75** (46.33, 52.21)	0.313 (0.265, 0.335)	8.188 (8.042, 8.273)	-1.02 (-1.03, -0.98)	5.393 (5.177, 5.745)	3.150 (1.708, 3.431)
SEARCH-GOTO	23.13 (22.90, 23.21)	**50.42** (48.27, 53.34)	0.311 (0.232, 0.337)	7.946 (7.853, 8.080)	-1.04 (-1.05, -1.02)	5.598 (5.272, 5.850)	2.584 (1.084, 3.219)
SEARCH-GOTOIF	**22.92** (22.61, 23.06)	**48.49** (44.61, 52.01)	0.283 (0.223, 0.336)	7.937 (7.844, 8.070)	-1.04 (-1.05, -1.01)	5.840 (5.624, 6.059)	2.283 (1.322, 3.028)

Table 4.10: SEARCH Series Architectures Fitness

Fitness results of the SEARCH series instruction set architectures. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations after sequential Bonferroni correction.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
SPLIT-IO	0.937 (0.934, 0.942)	0.694 (0.658, 0.719)	0.149 (0.148, 0.149)	0.449 (0.446, 0.463)	0.0 (0.0, 0.0)	0.927 (0.926, 0.930)	7.28×10^{-3} (3.99, 8.03)
SEARCH-DIRECTIONAL	**0.937** (0.932, 0.941)	0.626 (0.584, 0.652)	0.149 (0.148, 0.150)	0.448 (0.445, 0.455)	**0.0** (0.0, 0.0)	0.929 (0.927, 0.932)	7.67×10^{-3} (4.60, 8.01)
SEARCH-GOTO	0.940 (0.935, 0.943)	**0.644** (0.608, 0.676)	0.148 (0.147, 0.149)	0.447 (0.444, 0.452)	**0.0** (0.0, 0.0)	0.930 (0.928, 0.933)	6.43×10^{-3} (3.99, 7.61)
SEARCH-GOTOIF	**0.933** (0.928, 0.939)	**0.601** (0.569, 0.650)	0.149 (0.148, 0.150)	0.447 (0.445, 0.448)	**0.0** (0.0, 0.0)	0.929 (0.925, 0.933)	5.78×10^{-3} (4.15, 7.57)

Table 4.11: SEARCH Series Architectures Task Success

Task success results of the SEARCH series instruction set architectures. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries denote significant ($p < 0.05$, Wilcoxon rank-sum test) deviations.

4.1.3.6 Flow Control

The Flow series instruction sets tested three groups of flow control instructions separately and in several combinations (Tables 4.12 and 4.13). Throughout all instruction sets tested, the Fibonacci-32 environment showed no significant variation from the Search-Directional instruction set performance. The Match-12 environment had some significant drops in fitness, but these were not substantial and also not coupled with a drop in task success. The Logic-9 environment showed significant, though insubstantial, loss of fitness with all Flow series instruction sets. Three instruction sets, Flow-If0, Flow-IfX, and Flow-If0-IfX-MovHead, had corresponding small significant decreases in task success.

Individually, the If0 instruction group made virtually no difference in performance among any of the seven environments. When tested in combination with the other instruction groups, there is no clear indication of interaction, positive or negative.

The IfX instruction group both individually and in combination with other groups shows positive gains in the Navigation environment, both fitness and task success. This outcome is likely due to the nature of the signposts in this environment (Grabowski *et al.*, 2010), such that comparing against certain "magic" numbers for decision making is likely beneficial. The remaining six environments show no substantial variation attributable to these instructions.

The third instruction group, MovHead, shows the greatest variation in performance among those tested. In the Logic-77 environment, all instruction sets containing the MovHead group show substantial decreases in median fitness, 14.3% on average. The two combination sets containing MovHead, Flow-IfX-MovHead and Flow-If0-IfX-MovHead, also show corresponding decreases in task success in the Logic-77 environment. The Sort-10, Limited-9, and Navigation environments, on the other hand, show substantial improvements

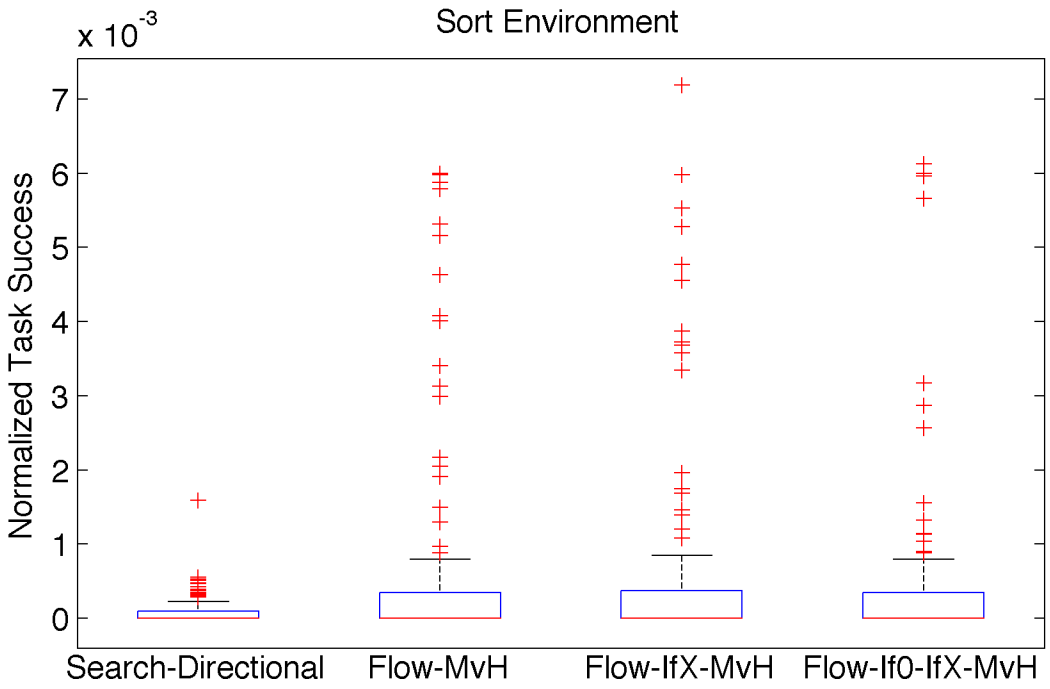

Figure 4.1: Selected FLOW Series Architectures Task Success Distribution
Normalized task success distributions of selected FLOW series instruction sets in the Sort-10 environment.

in task success, and often fitness, for all three instruction sets containing the MOVHEAD group. The Navigation environment, notably, approaches median task success around 1% when the IFX and MOVHEAD instruction groups are combined, indicating the importance of effective flow control for that environment. The Sort-10 environment improvements are difficult to observe from median values. Indeed the greatest driver of the improvements are infrequent outliers approaching 0.7% task success, the highest ever observed in the Sort-10 environment (see Figure 4.1).

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
SEARCH-DIRECTIONAL	23.14 (23.01, 23.23)	48.59 (46.46, 51.41)	0.313 (0.243, 0.346)	8.061 (7.990, 8.176)	-1.02 (-1.04, -1.01)	5.571 (5.334, 5.867)	3.022 (2.151, 3.387)
FLOW-MVH	**22.78** (22.38, 23.05)	**42.25** (39.76, 46.35)	0.277 (0.216, 0.347)	8.124 (7.990, 8.270)	**-0.96** (-1.01, -0.69)	5.474 (5.181, 5.771)	**3.875** (3.729, 4.052)
FLOW-IF0	**22.80** (22.62, 23.07)	47.99 (45.02, 51.50)	0.296 (0.258, 0.323)	7.995 (7.857, 8.099)	-1.04 (-1.05, -1.02)	5.553 (5.326, 5.855)	3.229 (2.919, 3.549)
FLOW-IFX	**22.64** (22.33, 22.86)	46.40 (44.51, 48.76)	**0.199** (0.168, 0.264)	8.037 (7.964, 8.198)	**-1.06** (-1.07, -1.03)	5.804 (5.460, 6.243)	**4.028** (3.861, 4.312)
FLOW-IF0-IFX	**22.82** (22.65, 23.01)	46.00 (42.64, 49.03)	**0.270** (0.201, 0.308)	8.011 (7.952, 8.078)	**-1.08** (-1.09, -1.07)	5.595 (5.292, 6.071)	**4.331** (4.113, 4.615)
FLOW-IFX-MVH	**22.95** (22.74, 23.11)	**41.29** (38.48, 44.64)	0.311 (0.244, 0.346)	8.063 (7.980, 8.193)	-1.00 (-1.03, -0.92)	5.751 (5.522, 6.061)	**4.475** (4.244, 4.983)
FLOW-IF0-IFX-MVH	**21.94** (21.55, 22.37)	**41.76** (39.12, 43.84)	**0.213** (0.189, 0.283)	7.995 (7.849, 8.066)	-1.01 (-1.05, -0.91)	6.077 (6.723, 6.625)	**5.036** (4.576, 6.457)

Table 4.12: FLOW Series Architectures Fitness

Fitness results of the FLOW series instruction set architectures. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations after sequential Bonferroni correction.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
SEARCH-DIRECTIONAL	0.943 (0.939, 0.946)	0.623 (0.584, 0.648)	0.148 (0.147, 0.149)	0.448 (0.444, 0.452)	0.0 (0.0, 0.0)	0.928 (0.926, 0.932)	7.55×10^{-3} (5.51, 8.04)
FLOW-MVH	0.938 (0.930, 0.946)	0.563 (0.532, 0.593)	0.150 (0.149, 0.150)	0.465 (0.452, 0.476)	**0.0** (0.0, 0.0)	**0.945** (0.941, 0.948)	$\mathbf{8.63\times10^{-3}}$ (8.40, 8.93)
FLOW-IF0	**0.932** (0.924, 0.937)	0.611 (0.581, 0.647)	0.148 (0.147, 0.149)	0.447 (0.446, 0.451)	0.0 (0.0, 0.0)	0.932 (0.930, 0.935)	7.78×10^{-3} (6.52, 8.12)
FLOW-IFX	**0.929** (0.916, 0.935)	0.607 (0.577, 0.622)	0.147 (0.146, 0.148)	0.447 (0.445, 0.453)	0.0 (0.0, 0.0)	0.931 (0.929, 0.933)	$\mathbf{8.88\times10^{-3}}$ (8.43, 9.67)
FLOW-IF0-IFX	0.937 (0.932, 0.941)	0.594 (0.550, 0.626)	0.148 (0.147, 0.149)	0.451 (0.446, 0.459)	0.0 (0.0, 0.0)	0.931 (0.928, 0.935)	$\mathbf{9.47\times10^{-3}}$ (8.75, 10.12)
FLOW-IFX-MVH	0.945 (0.940, 0.951)	**0.549** (0.508, 0.576)	**0.150** (0.149, 0.151)	0.459 (0.451, 0.467)	**0.0** (0.0, 0.0)	**0.941** (0.937, 0.947)	$\mathbf{9.65\times10^{-3}}$ (9.10, 10.64)
FLOW-IF0-IFX-MVH	**0.926** (0.901, 0.935)	**0.544** (0.516, 0.577)	0.150 (0.148, 0.150)	0.451 (0.447, 0.459)	**0.0** (0.0, 0.0)	**0.940** (0.935, 0.945)	$\mathbf{11.25\times10^{-3}}$ (10.23, 13.40)

Table 4.13: FLOW Series Architectures Task Success

Task success results of the FLOW series instruction set architectures. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries denote significant ($p < 0.05$, Wilcoxon rank-sum test) deviations.

4.1.4 Discussion

I have investigated the evolutionary potential of six groups of modified instruction set architectures of a digital evolution system, each within seven different computational environments. Among the groups investigated there were three classes of outcomes, broad multi-environment improvement, limited environment improvement, and no discernible trend. Notably absent from the observed classes was a broad detrimental change. Although some instruction set architectures did demonstrate decreased performance, there was only one example of highly substantial degradation, the Split-IO instruction set in the Sort-10 environment. I explore potential explanations for this particular case below. **In general, evolution has proven to be surprisingly robust to the explored genetic hardware changes, regardless of environment.**

Two groups of instruction set modifications yielded broadly beneficial changes in both fitness and task success. The Fully-Associative (FA) architectures instruction data flow enhancements led to highly significant gains in five of the seven environments. The remaining two environments, Sort-10 and Navigation, show some slight improvement and no discernible difference, respectively. The second group that demonstrated broadly positive results was the Split-IO instruction set. The separation of the input and output operation allows finer-grained data flow between the CPU and the environment. This control afforded by the Split-IO architecture was beneficial to the same five environments as the FA architecture. The Navigation environment showed no particular change in fitness performance, and a small, but insubstantial change in task success. The only major detriment to the splitting of input and output operations was observed in the Sort-10 environment. As a whole, these two groups indicate that it is beneficial to maintain as much flexibility as possible with regard to

instruction interactions. This flexibility allows evolution to finely tune interactions, yielding greater evolutionary potential.

The R series, Label series, and Search series architectures all demonstrated no discernible trend in performance, despite representing 17 of the 25 tested architectures. There were some particular environment/instruction set combinations that had significant variations, yet these were rarely substantial in nature. It is particularly surprising that the R series instruction sets showed such minimal deviation, given that going from the FA architecture to the R16 architecture represents a greater than five-fold increase in working set and a 50% increase in instruction set size. Similarly, the Label-Seq-Both instruction set represents a 20.6% increase in instruction set size, with no substantially negative effect. Taken together these groups hint at how robust the evolutionary process may be to genetic language dilution, maintaining the ability to adapt successfully to the environment despite searching a much larger genotype space. I explore this robustness in greater detail below in Section 4.3.

The Flow series of instruction set architectures represents a third class of outcomes, yielding improved results in a subset of environments and degradation of performance in one environment. The Sort-10, Limited-9, and Navigation environments all show substantial gains in both fitness and task success metrics when using instruction sets containing the IfX and MovHead instruction groups. The Logic-77 environment, on the other hand, shows a notable drop in performance. It is possible that this environment does not require a great deal of flow control, thus is being negatively affected by the disruptive nature of the additional flow control instructions. In environments where flow control decisions are critical for success, such as the Sort-10 and Navigation environment, the benefits of more flexible

flow control outweigh their disruptive effects.

The Sort-10 environment stands out as the only example where a single, small change, splitting the input and output instructions, made a large destructive difference in performance. Median task success collapsed to be statistically indistinguishable from 0, and remained there despite further beneficial instruction set modifications. These results are likely an artifact of the environment itself, rather than a general trend. I set up the Sort-10 environment to control for random inputs and to, on average, provide no benefit unless active sorting was performed by an organism. However, the inputs for sorting are indeed a random sample of 10 integers. It is possible, due to chance, for a partial ordering of numbers to yield a positive metabolic reward even if the sequence of inputs is simply echoed back to the environment. When using instruction sets featuring the paired-input-and-output instruction, simply mutating this instruction into the section of the genome responsible for replication may be enough to confer the echo capability, presenting an opportunity for lucky organisms to occasionally reap rewards. When the operations are split into two separate instructions, it then requires two coordinated mutations to confer the echo capability and doubles the execution cost for performing the task. The combination of these factors most likely contributes to the observed drop in median performance.

Instruction data flow, working set size, and flow control are the three main features addressed by the six groups of instruction set modifications presented here. All of these features play an important role in implementing a successful sorting algorithm. Despite the modifications in the instruction set architectures I tested, no significantly beneficial change was observed in either fitness or task success within the Sort-10 environment. Most likely, the highly constrained memory size of these architectures limits the potential within

this environment. In fact, a hand-written organism that performs the task successfully with the HEADS architecture requires nearly every single stack location in both available stacks. Another factor limiting potential may simply be the time allotted for evolution. The additional flow control instructions tested in the FLOW series architectures show some signs of improved success in this environment, with numerous outlier populations. Given additional time to evolve, these and other populations would likely be able to refine the emerging solutions.

When features from all six instruction set groups are combined to form the FLOW-IFX-MOVHEAD architecture, significant and substantial improvements relative to the base HEADS architecture are observed in six of the seven environments (Tables 4.14 and 4.15). Despite these improvements, there still remains a great deal of unexploited opportunity in five of the environments. Specific architectural changes to address these environments may yield greater results, such as the addition of an instruction capable of building arbitrary numbers for the Match-12 environment. However, such focused modifications could mask the need for more sweeping changes. Even with significant gains under two instruction sets, the Logic-77 environment still shows room for substantial improvement, as median task success shows populations utilizing less than 55% of the opportunities present. Even more so, the Sort-10 and Navigation environments exploit less than 1% of the available potential.

It is clear from this present study that we have just started to identify the most effective genetic hardware for adaptive evolution in digital organisms and there remains room for significant future improvement. I expect that further studies of instruction set architecture enhancements for evolvable systems will unlock this potential, facilitating advancements in the application of digital evolution and artificial life.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
HEADS	19.44 (17.74, 19.79)	13.50 (11.67, 15.30)	0.194 (0.168, 0.248)	3.453 (3.216, 3.858)	-0.47 (-0.61, -0.33)	4.328 (4.157, 4.445)	1.656 (1.108, 3.606)
FLOW-IFX-MVH	**22.95** (22.74, 23.11)	**41.292** (38.50, 44.56)	**0.311** (0.245, 0.347)	**8.063** (7.980, 8.189)	**-1.00** (-1.03, -0.92)	**5.751** (5.517, 6.049)	**4.475** (4.244, 4.953)

Table 4.14: HEADS and FLOW-IFX-MOVHEAD Architectures Fitness

Fitness results of the HEADS and FLOW-IFX-MOVHEAD instruction set architectures. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
HEADS	0.834 (0.752, 0.844)	0.185 (0.162, 0.211)	0.146 (0.145, 0.147)	0.202 (0.177, 0.228)	1.42×10^{-4} (1.08, 1.66)	0.908 (0.897, 0.914)	4.72×10^{-3} (3.99, 8.23)
FLOW-IFX-MVH	**0.945** (0.940, 0.951)	**0.549** (0.507, 0.577)	**0.150** (0.149, 0.151)	**0.459** (0.451, 0.467)	**0.0** (0.0, 0.0)	**0.941** (0.937, 0.947)	$\mathbf{9.65\times10^{-3}}$ (9.12, 10.62)

Table 4.15: HEADS and FLOW-IFX-MOVHEAD Architectures Task Success

Task success results of the HEADS and FLOW-IFX-MOVHEAD instruction set architectures. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries denote significant ($p < 0.05$, Wilcoxon rank-sum test) deviations.

4.2 Task Success Acquisition Rate of Selected Instruction Sets

The instruction set modifications explored in 4.1 demonstrated significant improvement in both fitness and task success over a broad range of architectures. The HEADS architecture, however, was already rather successful in two of the environments, Logic-9 and Limited-9, demonstrating 83.4% and 90.8% median task success after 100,000 updates, respectively. The changes in architecture tested above may have more substantial impact on evolutionary potential than can be observed from a single time point. Alterations to the architecture can affect both the rate of task success acquisition, in addition to the final achieved performance. In the discussion above (see Section 4.1.4), I also noted that environments may demonstrate greater overall task success given more evolutionary time. History plays a significant role in determining evolutionary potential (see Chapter 3). Architecture modifications that can limit the impact of history will continue to improve, while others stagnate. Here, I explore both of these aspects, examining the rate of task success acquisition and the longer term potential of selected instruction sets.

4.2.1 Methods

I tested the six instruction sets, the five reference architectures selected in Section 4.1, FA, R6, LABEL-SEQ-DIRECT, SEARCH-DIRECTIONAL, and FLOW-IFX-MOVHEAD, as well as the HEADS control architecture, in four environments that demonstrated the most substantial variation in performance, Logic-9, Logic-77, Fibonacci-32, and Limited-9. The architectures were tested using 50 replicate populations allowed to evolve for an extended time period of

	Logic-9		Logic-77		Fibonacci-32		Limited-9	
Heads	6700	(50)	201700	(16)	144500	(3)	6850	(50)
FA	4300	(50)	48900	(30)		(0)	5400	(50)
R6	4650	(50)	43900	(39)	41500	(7)	5600	(50)
Label-Seq-Direct	4700	(50)	38850	(38)	34400	(12)	5400	(50)
Search-Directional	5400	(50)	32400	(42)	16500	(18)	5250	(50)
Flow-IfX-MvH	4700	(50)	24500	(40)	93950	(26)	4600	(50)

Table 4.16: Time to 50% Task Success
Median time (in updates) to 50% task success of selected instruction set architectures. The number of replicates, out of 50 possible by 500,000 updates, that reached 50% task success is shown in parentheses.

500,000 updates. I collected data necessary for determining current population task success every 100 updates.

4.2.2 Results and Discussion

I used task success (Eq. 4.1), measuring the direct computational capabilities of the organisms in the population, to assess the rate at which each tested instruction set architecture evolves. For each replicate, I calculated the final population task success. If the final task success was greater than or equal to 50% of the theoretical maximum, I then determined the earliest update at which the population crossed the 50% threshold (with a granularity of 100 update intervals).

The results (Table 4.16) correspond well with the observations of Section 4.1.3. Each of the reference instruction sets, building upon the previous architecture, generally demonstrates reduced time to 50% task success and increased overall number of populations meeting that threshold. All replicate populations with all six tested instruction sets met the 50% threshold in the Logic-9 and Limited-9 environments. The FA instruction set demonstrates

the greatest improvement in those two environments, reducing time to 50% task success by 35.8% in Logic-9 and 21.2% in Limited-9. Median time to 50% task success remains largely consistent in both of those environments with the remaining four instruction sets. The Search-Directional instruction set, however, does slow evolution somewhat in the Logic-9 environment, while the Flow-IfX-MovHead instruction set improves Limited-9 performance by another 14.8%.

The Logic-77 environment demonstrates continuous improvement in both median time to 50% task success and number of populations meeting that threshold by the end of the experiment. The Flow-IfX-MovHead instruction set shows dramatic improvement compared to the Heads architecture, with 87.9% reduction in median time to 50% task success and a 2.5 fold increase in number of populations exceeding threshold.

The Fibonacci-32 environment shows a trend similar to the Logic-77, with continuous improvement. The Flow-IfX-MovHead instruction set does exhibit increased median time to 50% task success (93950 updates), relative to the Search-Directional architecture (16500 updates). However, the number of successful replicates is substantially increased as well, 26 with the Flow-IfX-MovHead architecture compared to 16 with the Search-Directional instruction set.

Final population performances (Tables 4.17 and 4.18) compare favorably to the results presented in Section 4.1.3. Given additional time, the longer term evolutionary potential of the R6, Label-Seq-Direct, Search-Directional, and Flow-IfX-MovHead instruction set architectures are essentially equivalent in the Logic-9, Logic-77, and Limited-9 environments. These results further support the observation in Section 4.1.4 that evolution is surprisingly robust to changes in architecture.

	Logic-9	Logic-77	Fibonacci-32	Limited-9
HEADS	23.54 (14.76, 23.88)	32.26 (11.43, 66.43)	5.562 (2.952, 21.98)	4.184 (3.098, 15.31)
FA	23.54 (14.79, 23.89)	**44.42** (15.79, 72.85)	**6.595** (4.166, 8.45)	**4.665** (3.077, 10.99)
R6	23.48 (14.70, 23.85)	**57.68** (21.93, 73.09)	6.263 (3.035, 21.93)	**4.841** (3.079, 10.04)
LABEL-SEQ-DIRECT	23.32 (14.08, 23.82)	**57.38** (19.14, 72.96)	**7.555** (3.801, 22.12)	**5.089** (2.898, 11.11)
SEARCH-DIRECTIONAL	23.55 (17.84, 23.81)	**55.77** (19.38, 72.68)	**8.518** (7.577, 22.22)	**5.028** (3.111, 22.13)
FLOW-IFX-MVH	23.32 (13.96, 23.81)	**57.94** (18.28, 73.55)	**8.706** (7.701, 22.64)	**5.736** (2.243, 18.36)

Table 4.17: Extended Architecture Experiments Fitness
Final fitness results of selected instruction set architectures run for extended experiments. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations.

Final population performance in the Limited-9 environment does demonstrate a minor, though not significant trend of increasing fitness and task success with increasing instruction set complexity. The Fibonacci-32 environment, however, continues to show notably improved evolutionary potential, indicating that the architectural changes remain beneficial to some environments, despite making no significant difference in the other three tested environments. Although, as previously mentioned, the no free lunch theorem states that no "perfect" encoding exists, these results support my hypothesis that the current genetic hardware in Avida is far from optimal and that substantial, generic improvements in performance appear to be possible.

	Logic-9	Logic-77	Fibonacci-32	Limited-9
HEADS	0.936 (0.743, 0.966)	0.418 (0.158, 0.800)	0.301 (0.167, 0.969)	0.920 (0.820, 0.952)
FA	**0.949** (0.744, 0.975)	**0.558** (0.215, 0.906)	**0.350** (0.224, 0.464)	0.928 (0.883, 0.966)
R6	0.946 (0.744, 0.970)	**0.734** (0.292, 0.921)	0.325 (0.171, 0.944)	**0.934** (0.905, 0.967)
LABEL-SEQ-DIRECT	0.943 (0.731, 0.969)	**0.714** (0.266, 0.925)	**0.435** (0.191, 0.950)	**0.933** (0.906, 0.966)
SEARCH-DIRECTIONAL	**0.951** (0.804, 0.977)	**0.722** (0.263, 0.915)	**0.472** (0.400, 0.963)	**0.932** (0.879, 0.977)
FLOW-IFX-MVH	**0.958** (0.746, 0.986)	**0.753** (0.238, 0.947)	**0.505** (0.443, 0.984)	**0.954** (0.750, 0.985)

Table 4.18: Extended Architecture Experiments Task Success
Final normalized task success results of selected instruction set architectures run for extended experiments. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations.

4.3 Assessing Robustness to Poor Design Decisions

Note, the material in the section previously appeared as Bryson and Ofria (2012).

Changes to the instruction set can have a profound effect on the evolutionary potential of the system with respect to the environment, as shown above in Section 4.1.3. It is difficult, if not impossible, to asses the impact of instruction set changes *a priori*. A seemingly beneficial change may, in fact, have unintended negative interactions with other aspects of the system. Here I describe results of investigations into three types of poor instruction set design decisions: functionally neutral instructions that bloat the instruction set, actively deleterious instructions that poison the organism, and a lethal instruction. I evaluate the evolutionary potential of each instruction set using the same approach as Section 4.1.

Given a fixed environment, one particular instruction set may show greater evolutionary

potential in comparison to another. However, it is possible that aspects of an instruction set may demonstrate better adaptability to changing circumstances. As evolution progresses, genomes lock in features and genetic organization that are beneficial. The structure of these genomes impact their potential when the environment changes. I investigate this connection by evaluating short term evolutionary potential of genomes following environmental change. I examine how this potential changes relative to the progress of evolution in the origin environment.

4.3.1 Tested Instruction Set Modifications

The BLOAT-X instruction sets test what happens if you add useless, although not directly disruptive, instructions to the instruction set. They extend the 26 instructions in the HEADS architecture with the addition of one or more copies of the no-operation instruction, `nop-X`, which is functionally neutral, in that it does not alter the state of the virtual CPU. Additionally, unlike the default no-operation instructions, it does not alter the behavior of other instructions. We tested four BLOAT-X instruction sets, varying the mutational frequency of the `nop-X` instruction. BLOAT-1 adds `nop-X` with a frequency of one, increasing the overall instruction set size to 27 and yielding an effective mutational frequency of 0.037 for each instruction. BLOAT-3, BLOAT-10, and BLOAT-30 increase the frequency of `nop-X` to 3, 10, and 30, respectively. In BLOAT-30, the effective instruction set size is 56, with an effective mutational frequency of the `nop-X` instruction at 0.536, and 0.018 for each of the remaining 26 standard instructions.

With the POISON instruction sets we are testing what happens when we make a poor decision by adding an instruction that disrupts the functionality of the organisms upon

execution. These instruction sets extend the HEADS architecture with the addition of a `poison` instruction that, when executed, reduces the metabolic rate of the organism by a configurable severity. Reduced metabolic rate translates to fewer relative CPU cycles, and therefore diminished competitive ability. We tested three poison severities, 0.003, 0.01, 0.03, which reduce metabolic rate by 0.3%, 1%, and 3%, respectively, each time the organism executes the instruction. We hypothesized that lower penalties might be more detrimental to long-term evolution, because they may slip in and accumulate over time.

Lastly, the DIE instruction set sought to determine what happens when we make a catastrophic error in including an instruction in the set. This instruction set adds a single `die` instruction to the HEADS architecture. The presence of a `die` instruction in a genome is not itself lethal. If the organism executes the instruction during its' lifetime, however, the organism will be immediately removed from the population. In this manner, the `die` instruction is similar to a `poison` instruction with a severity of 1.0.

4.3.2 General Performance Evaluation

As described in Section 4.1.3, I have focused on two measures of evolved populations to evaluate the general performance of each instruction set: mean fitness and task success. Both measure the ability of the evolved organisms to perform tasks within the environment.

Upon analysis, all three POISON instruction sets and the DIE instruction set demonstrated no significant variation in either population mean fitness (Table 4.19) or task success (Table 4.20) across all seven environments. Indeed, the distributions of observed results were largely similar, regardless of the severity of the penalty associated with a given instruction. Likewise, the BLOAT-1 and BLOAT-3 instructions sets showed comparable performance to

the HEADS control. This result would indicate that a single poor choice of an instruction, no matter how bad, is not likely to significantly limit evolutionary outcomes.

The BLOAT-10 and BLOAT-30 instruction sets, representing high levels of instruction mutational dilution, demonstrate some negative effects in final population performance. In the Logic-9 environment, both BLOAT-10 and BLOAT-30 showed significantly decreased fitness and task success. Runs with these instruction sets evolved one fewer task on average in the Logic-9 environment when compared to the control. The Logic-77 environment similarly demonstrated decreased fitness and task success, but unlike in the Logic-9 environment, the BLOAT-30 instruction set was notably worse than BLOAT-10. This pattern of declining performance was also observed in the Fibonacci-32 environment, with task success indicating that the populations are outputting fewer one to three fewer numbers in the sequence as instruction set dilution increases.

The Limited-9 environment demonstrated a split between fitness and task success results for BLOAT-10 and BLOAT-30. The fitness results for both instruction sets were greater than the HEADS control, though not significantly after Bonferroni correction. The BLOAT-10 instruction set demonstrated task success that was somewhat, though not significantly, reduced compared to the control. Task success with the BLOAT-30 instruction set, however, was significantly reduced, with populations typically evolving one fewer task, as compared to the HEADS instruction set.

In the Navigation environment, the BLOAT-10 instruction set demonstrated comparable performance to the control for both fitness and task success. The BLOAT-30 instruction, on the other hand, showed significantly improved median fitness. Task success was also notably increased, nearly double all other instructions sets, though not statistically significant from

the control after Bonferroni correction. Despite the increase, the populations are still quite far from exploiting the opportunities in this environment, taking advantage of less than 1% of the potential resources.

The Match-12 environment showed no variation in task success with any of the tested instruction sets. The BLOAT-10 and BLOAT-30 instruction sets both demonstrated significantly lower fitness, with BLOAT-30 the most severely depressed. Given the lack of variation in task success, these fitness result likely reflect the impact of neutral instruction set bloat on the evolution of replication efficiency in these digital organisms.

Lastly, the Sort-10 environment was significantly reduced in both fitness and task success under both BLOAT-10 and BLOAT-30. The differences observed, however, were relatively insubstantial. None of the tested instruction sets, including the HEADS control, were able to take advantage of the opportunities in the Sort-10 environment; all sets demonstrated $\ll 1\%$ of the potential task success. The current limitations of the virtual CPU appear to make this task incredibly difficult to evolve.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
HEADS	19.71 (19.33, 19.81)	15.22 (12.81, 16.34)	0.198 (0.159, 0.243)	3.645 (3.299, 4.082)	-0.482 (-0.598, -0.324)	4.220 (4.095, 4.294)	1.447 (1.103, 3.197)
BLOAT-1	19.33 (18.12, 19.72)	14.42 (12.36, 15.89)	0.211 (0.180, 0.242)	3.241 (3.010, 3.815)	-0.498 (-0.608, -0.372)	4.301 (4.141, 4.418)	1.123 (1.051, 2.453)
BLOAT-3	19.67 (18.15, 19.75)	12.31 (10.71, 14.37)	0.176 (0.142, 0.207)	3.400 (3.206, 3.762)	-0.538 (-0.611, -0.441)	4.247 (4.138, 4.375)	1.123 (1.031, 3.202)
BLOAT-10	**14.80** (14.73, 17.32)	**11.77** (10.66, 14.32)	**0.106** (0.082, 0.127)	**2.621** (2.540, 3.116)	**-0.696** (-0.717, -0.675)	4.526 (4.312, 4.779)	1.068 (1.022, 2.640)
BLOAT-30	**14.38** (13.70, 14.61)	**7.74** (7.69, 8.67)	**-0.053** (-0.170, 0.083)	**1.768** (1.722, 1.802)	**-0.770** (-0.785, -0.741)	4.480 (4.317, 4.684)	**2.872** (1.313, 3.165)
POISON-0.003	19.57 (18.72, 19.73)	14.11 (12.19, 15.96)	0.206 (0.177, 0.252)	4.240 (3.496, 4.623)	-0.342 (-0.483, -0.225)	4.152 (4.071, 4.226)	1.157 (1.056, 1.635)
POISON-0.01	19.41 (18.20, 19.76)	12.63 (11.64, 14.76)	0.159 (0.137, 0.214)	3.309 (3.181, 3.825)	-0.591 (-0.664, -0.442)	4.300 (4.127, 4.492)	1.342 (1.052, 3.386)
POISON-0.03	18.66 (17.79, 19.62)	12.19 (11.42, 14.40)	0.157 (0.130, 0.192)	3.417 (3.219, 3.883)	-0.464 (-0.616, -0.313)	4.290 (4.207, 4.538)	1.356 (1.068, 3.275)
DIE	19.60 (17.84, 19.78)	14.28 (11.93, 16.32)	0.181 (0.156, 0.235)	3.476 (3.211, 3.909)	-0.581 (-0.634, -0.438)	4.422 (4.288, 4.563)	1.324 (1.060, 3.150)

Table 4.19: BLOAT-X, POISON-X, and DIE Architectures Fitness

Fitness results for all 8 test instruction sets and the HEADS control architecture. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations after sequential Bonferroni correction.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
HEADS	0.842 (0.835, 0.847)	0.207 (0.179, 0.227)	0.145 (0.144, 0.146)	0.205 (0.177, 0.237)	1.42×10^{-4} (1.14, 1.71)	0.906 (0.896, 0.912)	4.35×10^{-3} (3.98, 7.54)
BLOAT-1	0.835 (0.753, 0.843)	0.198 (0.173, 0.218)	0.146 (0.145, 0.147)	0.177 (0.174, 0.207)	1.46×10^{-4} (1.10, 1.67)	0.906 (0.896, 0.911)	4.05×10^{-3} (3.96, 5.84)
BLOAT-3	0.839 (0.825, 0.846)	0.171 (0.150, 0.197)	0.146 (0.145, 0.147)	0.203 (0.176, 0.236)	1.37×10^{-4} (1.14, 1.54)	0.899 (0.829, 0.911)	3.99×10^{-3} (3.97, 7.20)
BLOAT-10	**0.747** (0.744, 0.750)	0.166 (0.150, 0.203)	0.146 (0.144, 0.146)	**0.174** (0.149, 0.178)	$\mathbf{1.01\times10^{-4}}$ (0.99, 1.04)	0.832 (0.823, 0.894)	4.00×10^{-3} (3.97, 6.86)
BLOAT-30	**0.736** (0.648, 0.744)	**0.114** (0.113, 0.126)	0.146 (0.125, 0.147)	**0.120** (0.119, 0.120)	$\mathbf{9.7\times10^{-5}}$ (9.6, 9.9)	**0.777** (0.732, 0.808)	7.57×10^{-3} (4.08, 7.82)
POISON-0.003	0.841 (0.828, 0.846)	0.194 (0.172, 0.217)	0.147 (0.146, 0.148)	0.239 (0.205, 0.285)	1.67×10^{-4} (1.45, 1.97)	0.910 (0.903, 0.915)	3.99×10^{-3} (3.97, 4.79)
POISON-0.01	0.839 (0.821, 0.845)	0.174 (0.162, 0.204)	0.146 (0.145, 0.146)	0.179 (0.176, 0.208)	1.22×10^{-4} (1.07, 1.54)	0.898 (0.844, 0.909)	4.33×10^{-3} (3.97, 7.77)
POISON-0.03	0.838 (0.773, 0.844)	0.170 (0.159, 0.202)	0.146 (0.145, 0.147)	0.202 (0.177, 0.237)	1.52×10^{-4} (1.11, 1.82)	0.911 (0.897, 0.916)	4.33×10^{-3} (3.97, 7.97)
DIE	0.827 (0.754, 0.841)	0.198 (0.163, 0.224)	0.146 (0.145, 0.147)	0.195 (0.176, 0.210)	1.27×10^{-4} (1.04, 1.52)	0.906 (0.840, 0.913)	4.29×10^{-3} (3.97, 6.58)

Table 4.20: BLOAT-X, POISON-X, and DIE Architectures Task Success

Task success results for all 8 test instruction sets and the HEADS control architecture. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$, Wilcoxon rank-sum test) deviations after sequential Bonferroni correction.

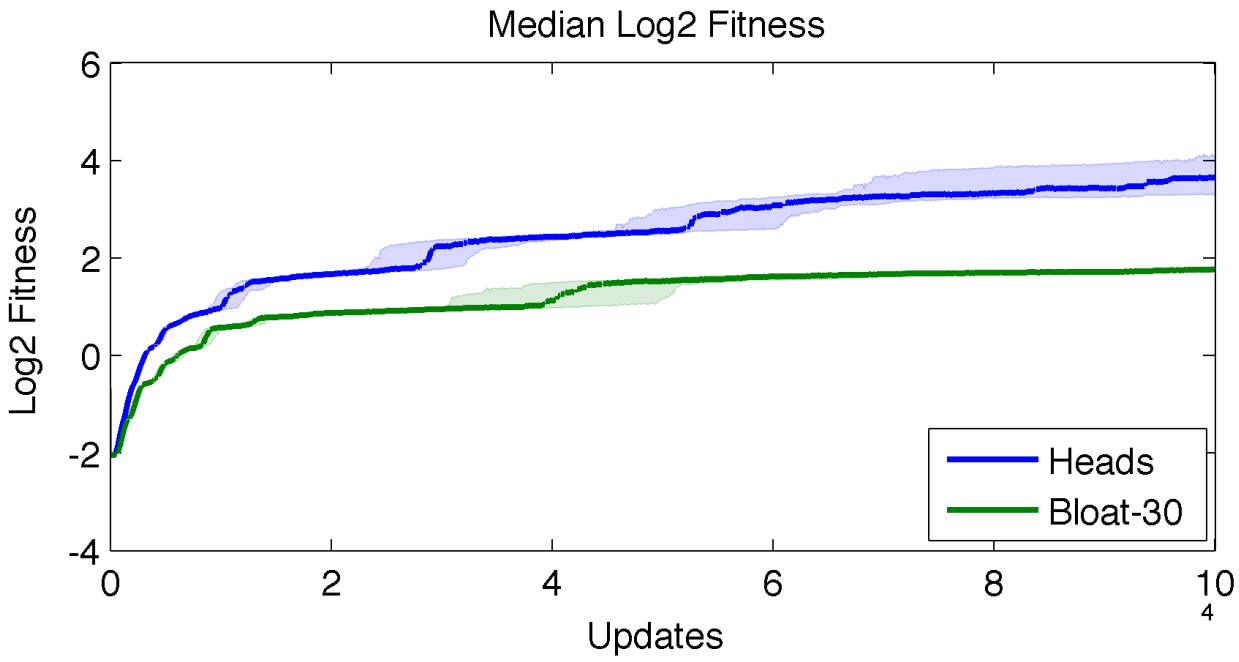

Figure 4.2: HEADS and BLOAT-30 Fitness Trajectory
Median $\log_2$ fitness trajectory of the HEADS (blue line) and BLOAT-30 (green line) architectures. Lines calculated from all 200 replicates of each instruction set in the Fibonacci-32 environment. Shaded regions show 95% bootstrap confidence intervals, 10,000 iterations.

4.3.3 Impact on Evolutionary Potential in a New Environment

The BLOAT-X instruction sets, especially BLOAT-10 and BLOAT-30, showed reduced performance when examining a fixed end point, as shown above. The fitness trajectories of these instruction sets demonstrated a corresponding drag on evolution throughout the entire history of the runs, relative to the HEADS control instruction set (see Figure 4.2 for an example). Despite this apparent drag, fitness was still rising at the end of the experiment, albeit more slowly.

The neutral bloat represented by the BLOAT-X instruction sets, although detrimental to the rate of evolution, may have a beneficial effect on the genetic architecture of the evolved genomes. The nop instructions tend to decouple strings of other instructions, such that genetic functions must be more loosely coupled. This property may afford greater

evolutionary potential when the genomes experience environmental change. We have tested this hypothesis by performing STEP sampling of genotypes in a new environment, never before encountered in the history of the genotype.

We extracted the principal line of descent from 12 selected runs utilizing the HEADS and BLOAT-30 instruction sets that we initially evolved in the Logic-9 and Fibonacci-32 environments. These two environments both demonstrated variation in performance between the HEADS and BLOAT-30 instruction sets, and present computationally unique challenges to the organisms (logic computation in Logic-9 and loop coordination in Fibonacci-32). The genotypes along each of the lines of descent were STEP sampled, still using their native instruction set, but in the opposite environment. For example, we placed genotypes evolved in the Logic-9 environment into the Fibonacci-32 environment and evolved them for 10,000 updates. We sampled each genotype ten times and examined the fitness and task success of the resulting populations.

The STEP sampling results of the HEADS control instruction set show that the short term evolutionary potential of the genotypes declined in the Logic-9 environment as evolution progressed in the Fibonacci-32 environment (see Figure 4.3). All sampled lines of descent with the HEADS architecture demonstrate similar patterns of evolutionary potential in the Fibonacci-32 to Logic-9 shift. The BLOAT-30 instruction set runs, on the other hand, show a relatively flat trend of evolutionary potential. Additionally, the BLOAT-30 instruction set demonstrated an increased number of high potential outliers throughout all of the sampled lines of descent originally evolved in the Fibonacci-32 environment (see Figure 4.5).

Both the HEADS control and the BLOAT-30 demonstrated a consistent pattern of gradual decline in short term evolutionary potential when sampling genotypes originally evolved in

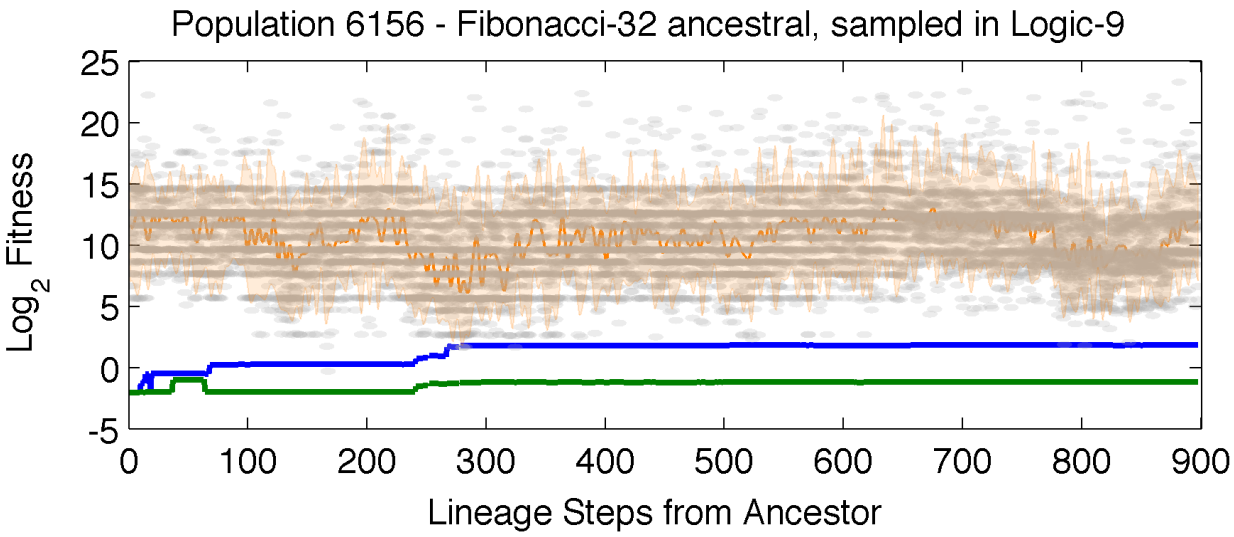

Figure 4.3: STEP Sampling of Population 6156
STEP sampling results of population 6156 originally evolved in the Fibonacci-32 environment with the HEADS instruction set. Blue line: fitness of the reference genotype in its native Fibonacci-32 environment. Green line: initial fitness of the reference genotype in the alternative Logic-9 environment. Orange line: median STEP results, smoothed using a FFT, shown with 95% quantiles. Gray circles: individual STEP results. The first point (step 0) shows the performance of the ancestral genotype.

the Logic-9 source environment within the Fibonacci-32 sample environment (see Figure 4.4). Similar to the Fibonacci-32 to Logic-9 environment transition, the BLOAT-30 instruction set showed notably more outlier samples of high potential. However, the overall spread and trend of samples of the BLOAT-30 genotypes were comparable to the HEADS instruction set.

In order to assess the generality of the observed patterns, we STEP sampled the final dominant genotype from all 200 runs of each of the original HEADS and BLOAT-30 experiments from the Logic-9 and Fibonacci-32 environments in the appropriate alternate environment. As observed in the line of descent sampling, the BLOAT-30 instruction set genotypes from the Fibonacci-32 environment demonstrated significantly greater potential ($p < 0.017$; Wilcoxon rank-sum test) when sampled in the Logic-9 environment (median $\log_2$ fitness 10.40) in comparison to genotypes evolved with the HEADS instruction set (median $\log_2$ fitness 9.653).

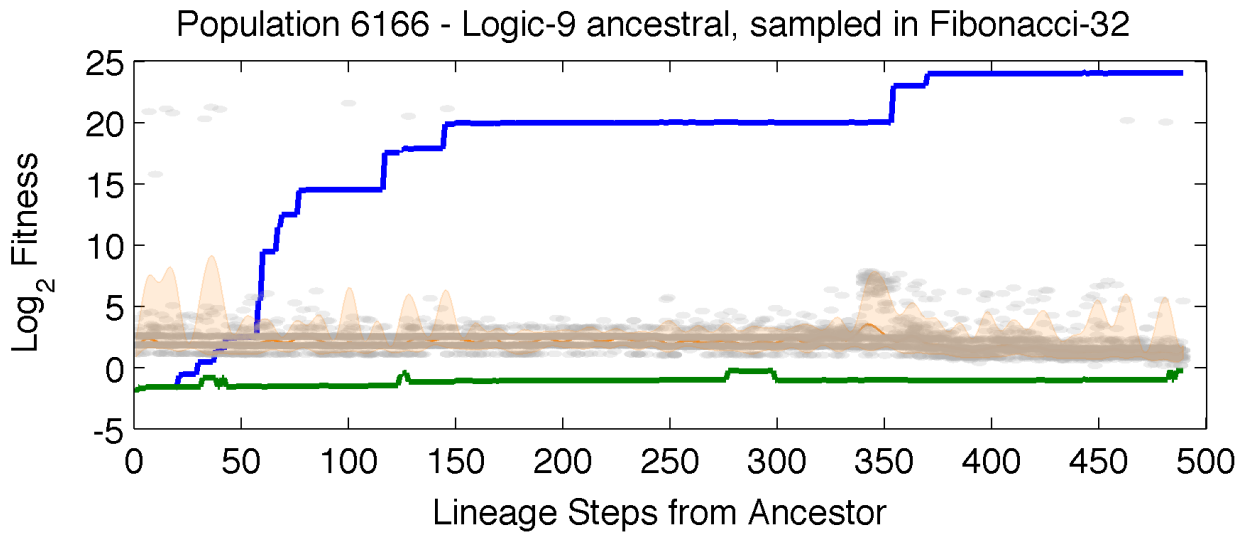

Figure 4.4: STEP Sampling of Population 6166

STEP sampling results of population 6166 originally evolved in the Logic-9 environment with the HEADS instruction set. Blue line: fitness in its native Logic-9 environment. Green line: initial fitness in the alternative Fibonacci-32 environment. Orange line: median STEP results (smoothed) with 95% quantiles. Gray circles: individual STEP results.

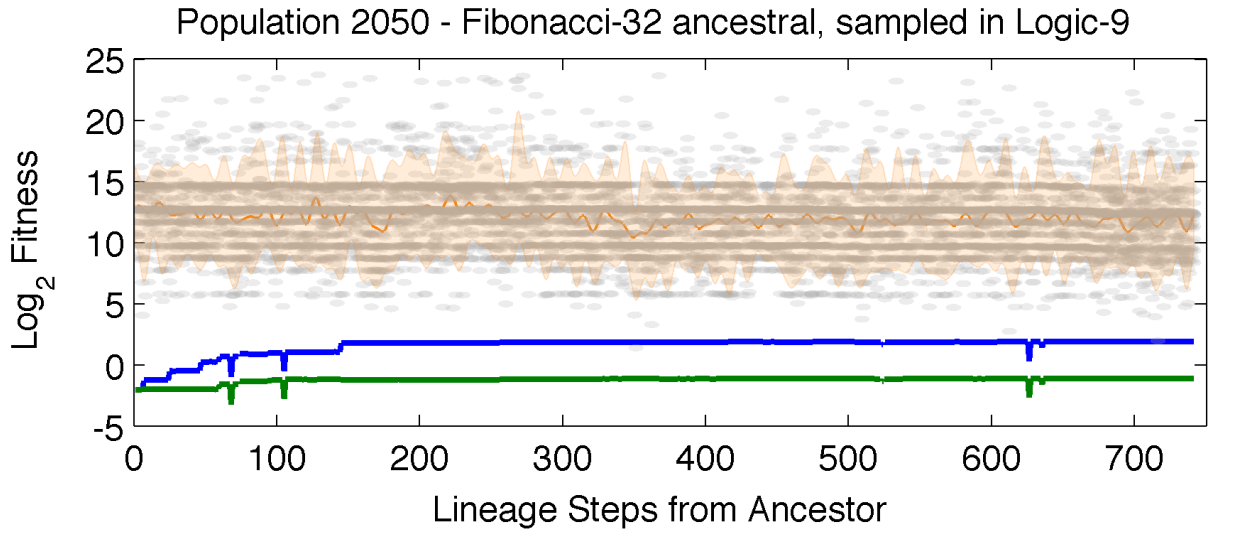

Figure 4.5: STEP Sampling of Population 2050

Fitness STEP sampling results of population 2050 originally evolved in the Logic-9 environment with the HEADS instruction set. Blue line: fitness in its native Logic-9 environment. Green line: initial fitness in the alternative Fibonacci-32 environment. Orange line: median STEP results (smoothed) with 95% quantiles. Gray circles: individual STEP results.

The transition from the Logic-9 environment to the Fibonacci-32 environment showed the opposite results, with the HEADS instruction set resulting in significantly greater ($p < 0.026$) evolutionary potential (median $\log_2$ fitness 1.836; Wilcoxon rank-sum test) in comparison to the BLOAT-30 instruction set (median $\log_2$ fitness 1.768).

4.3.4 Discussion

In examination of general performance, evolution demonstrated surprising robustness to increasingly poor design decisions. The addition of individual instructions that were incredibly deleterious or lethal made no significant difference in the evolutionary potential of the system across a wide range of static test environments. Similarly, low levels of neutral instruction set bloat contributed negligibly to the observed performance. These results indicate that digital evolution can reasonably overcome individual or small sets of detrimental design decisions, regardless of the severity of the error. Populations exhibit substantial declines only if many poor decisions compound one another.

High levels of instruction set bloat, diluting the frequency of functional mutations, resulted in an overall significant drag on evolution. Despite this dilution decreasing the rate of evolution, populations were still gaining fitness and task success, indicating that evolution could potentially overcome the detrimental effects of such poor designs given additional time. Although BLOAT-30 performed poorly in the initial experiments, STEP sampling showed that, under certain circumstances, the increased proportion of neutral mutations associated with instruction set bloat can actually improve evolutionary potential when changing the environment. The genetic architecture of the genotypes from the Fibonacci-32 environment with the BLOAT-30 instruction set, broken up by neutral instructions, showed to be more

adaptable to the logic flow necessary for success in the Logic-9 environment. Conversely, the genotypes with the BLOAT-30 instruction set evolved in the Logic-9 environment performed worse in the Fibonacci-32 environment, indicating that the looping structures necessary in the Fibonacci-32 environment likely benefit from more closely connected instructions.

4.4 Future Work

There are many potential avenues for further exploration of genetic hardware beyond the scope of this dissertation. Among those avenues is performing head-to-head competition experiments between instruction sets to directly assess the evolutionary potential of particular architectures. Additionally, there are an almost infinite number of additional architecture modifications that are worth investigating, including multiple memory spaces, changing registers to stacks, implementing instructions to facilitate the generation of arbitrary numbers, splitting the h-copy instruction into read and write instructions, adding threading capabilities, implementing an interrupt-driven execution to facilitate environmental interaction, and developing gene-regulatory-network inspired flow control.

Chapter 5

How Do Mutation Types Affect Evolutionary Potential?

A source of variation is a required component for adaptive evolution. Mutation rates clearly affect the rate of adaptive evolution, but are some types of mutations better than others? Mutations can take on many forms and may occur at multiple levels of activity in a system. Genotypic mutations take place in the heritable substrate, producing lasting variation. In contrast, phenotypic mutations occur during expression of the genotype, producing transient variation. Such transient variation, however, may trigger other types of genotypic variation.

In addition to type, mutations can also vary in their rate, effect size, target, and distribution. Mutations can occur during an organism's lifetime or only during replication. The probability of mutation occurring may be fixed, or it may be dependent on other factors, such as genome size, organism activity, location, or simply time. The effect size of a mutation may impact only a single position in a genome (be it a nucleotide, an amino acid, or an instruction) or a mutation may change many elements simultaneously, such as by duplicat-

ing subroutines, genes, chromosomes, or even whole genomes. Mutations may target specific elements or may be general rates across all such components. The distribution of characters or activities that may be substituted as the result of a mutation can be uniform, skewed, or dependent upon the original element being mutated.

Clearly, there are many aspects of mutations that can be varied, and it is unclear *a priori* what type of role each will play in terms of evolutionary potential. While the genotype-phenotype relationship may set the characteristics of each point in a fitness landscape, these mutational rules determine its connectivity. Certain genotypic mutations may allow evolution to traverse rugged regions of the landscape that would otherwise have been nearly impassable. Others, however, may force the evolution of greater genotypic robustness, at the cost of adaptive ability.

The field of evolutionary computation, seeking to optimize search performance, has explored many mutation types (which are called "search operators" in the field), typically varying many of the characteristics described above. Given the focus on solving engineering problems, evaluation of new search operators typically focuses on short-term problem solving capability. In biological and digital evolution, additional properties, such as robustness, modularity, adaptive complexity, and future evolvability are also important. To date there has been no systematic exploration of mutation types and their effect on evolutionary potential in an open-ended evolution system. I have explored four main aspects of mutations, the optimal balance of mutation types (Section 5.2), mutation timing (Section 5.3), versions of large insertion/deletion mutations (Section 5.4), and the side-effects of mutations (Section 5.5). Each treatment was tested in the seven computational environments described in Section 2.2, evaluating how these mutations alter evolutionary potential.

5.1 Methods

I have evaluated the evolutionary potential of all mutation types described below using sets of 50 replicate populations run for 500,000 updates. During these runs I collected two metrics for comparison, mean population fitness and normalized task success, every 100 updates. Normalized task success, described in detail in Section 4.1.3, is a standardized measure of the environmental task performance of a genotype.

The overall mutation rate of 3×10^{-3} per site in the genome identified as the most broadly beneficial in Section 5.2.1 was using in all subsequent experiments in this chapter. Additionally, the balanced proportion (1:1) of insertion/deletion mutations and instruction substitution mutations found to be the most broadly beneficial in Section 5.2.2 was used in all subsequent experiments in this chapter.

5.2 Exploring the Balance of Mutation Types

The standard mutation types utilized in Avida are single instruction substitution, and single instruction insertion or deletion. These rates and proportions of these mutations have been selected based on previous work in limited environments. I have systematically explored the balance of mutation rates, proportions, and probability applications in all seven computational environments, assessing the evolutionary potential of the system given a fixed time period.

5.2.1 Optimal Mutation Rates for Adaptive Evolution

Mutation rate plays a critical role in determining the speed of adaptive evolution. High rates allow rapid exploration of a fitness landscape, but if the rates are too high adaptive evolution will slow down as populations struggle to maintain any genetic information.

I tested five mutation loads, 0.01, 0.1, 0.3, 1.0, and 3.0, such that given the genome length of the default ancestral genome the 1.0 rate would on average yield one mutation per generation. The results, presented in Tables 5.1 and 5.2, match well with expectations, with most environments demonstrating the greatest evolutionary potential at intermediate mutation rates. The Logic-9, Match-12, Sort-10, and Navigation environments all show the highest fitness and generally task success with 0.3. The Logic-77 and Fibonacci-32 environments appear to peak with the 1.0 rate. The Limited-9 environment demonstrates an unexpected tradeoff between fitness and task success. Median fitness continued to increase with increased mutation rate, even at 3.0. However, task success appears to to peak around 0.1, with steady decline in performance as mutation rate increases.

As indicated in the methods, above, I selected the 0.3 mutation rate to perform all of the other experiments in this Chapter.

Rate	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
0.01	9.77 (2.00, 17.41)	4.88 (0.71, 9.66)	-0.086 (-0.423, 0.55)	1.342 (0.541, 4.44)	-0.551 (-1.211, -0.274)	1.861 (-0.099, 5.996)	0.574 (-0.148, 4.208)
0.1	19.89 (12.71, 24.10)	12.54 (6.36, 63.81)	0.336 (-0.154, 0.59)	3.462 (1.414, 18.82)	-0.377 (-0.934, 0.906)	3.745 (2.642, 7.253)	3.956 (1.101, 18.475)
0.3	23.63 (14.75, 23.94)	28.20 (10.18, 72.29)	0.366 (0.137, 0.76)	5.736 (2.572, 21.67)	0.126 (-0.736, 1.130)	4.354 (2.894, 15.336)	5.358 (1.033, 22.181)
1.0	23.04 (14.46, 23.48)	45.75 (11.81, 68.62)	-0.003 (-0.461, 0.28)	7.659 (2.755, 21.65)	-0.427 (-0.906, 1.940)	4.364 (2.830, 12.115)	1.227 (1.048, 21.070)
3.0	17.24 (12.02, 22.06)	48.63 (8.03, 70.37)	-0.800 (-0.917, -0.76)	6.007 (3.335, 20.60)	-0.799 (-0.922, -0.762)	7.882 (2.049, 15.370)	0.791 (0.638, 3.575)

Table 5.1: Mutation Rate Fitness

Fitness results of tested mutation rates. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses.

Rate	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
0.01	0.555 (0.256, 0.815)	0.078 (0.021, 0.145)	0.083 (0.074, 0.166)	0.094 (0.062, 0.322)	0.99×10^{-4} (0.00, 1.48)	0.630 (0.269, 0.885)	2.805×10^{-3} (2.045, 9.361)
0.1	0.871 (0.639, 0.988)	0.177 (0.093, 0.846)	0.160 (0.079, 0.163)	0.184 (0.093, 0.990)	1.42×10^{-4} (0.92, 5.20)	0.970 (0.709, 0.982)	8.713×10^{-3} (3.671, 27.646)
0.3	0.947 (0.740, 0.970)	0.370 (0.143, 0.903)	0.151 (0.144, 0.221)	0.307 (0.148, 0.912)	2.62×10^{-4} (0.93, 6.05)	0.931 (0.815, 0.958)	10.293×10^{-3} (3.938, 42.982)
1.0	0.876 (0.702, 0.926)	0.512 (0.156, 0.834)	0.122 (0.063, 0.138)	0.423 (0.152, 0.910)	1.46×10^{-4} (0.00, 13.49)	0.845 (0.714, 0.925)	3.880×10^{-3} (3.355, 40.114)
3.0	0.687 (0.539, 0.892)	0.508 (0.121, 0.735)	0.000 (0.000, 0.000)	0.350 (0.207, 0.817)	0.00×10^{-4} (0.00, 0.00)	0.729 (0.451, 0.891)	3.491×10^{-3} (3.298, 8.881)

Table 5.2: Mutation Rate Task Success

Normalized task success results of tested mutation rates. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses.

5.2.2 Balance of Insertion/Deletions and Substitution Mutations

In the default configuration, insertion/deletion (indel) mutations and substitutions occur with proportions 2:5, values that were originally chosen in an arbitrary manner. I have varied these proportions with seven treatments, increasing or decreasing insertion/deletion mutations relative to substitution mutations. Specifically, I tested the proportions 0:1 (substitutions only), 1:5, 2:5, 1:1, 5:2, 5:1, 1:0 (indels only). The 0:1 treatment group, featuring only substitution mutations, did not require organisms to maintain fixed length, but length changes required unstable genotypes, a side-effect of other mutations to be discussed in detail in Section 5.5.

Analysis of the results (see Tables 5.3 and 5.4) showed that five of the tested proportions, 1:5, 1:1, 5:2, 5:1, 1:0 (indels only), were, for all practical purposes, equivalent in performance. The 0:1 (substitutions only) treatment was substantially worse in the Logic-9, Logic-77, Fibonacci-32, and Navigation environments, likely due to the fixed length aspect of the treatment. The 2:5 treatment, representing the default setting for Avida, demonstrated a notable tradeoff between the Logic-77 and Fibonacci-32 environments, where median fitness and task success were the highest of all treatments, and the Match-12, Sort-10, and Navigation environments, where fitness and task success were substantially depressed relative to all other treatments.

As indicated in the methods, above, I selected the 1:1 proportion of indel:substitution mutations for all other experiments in this Chapter.

Proportion	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
0:1	16.61 (9.76, 24.04)	11.78 (4.26, 60.85)	0.365 (-0.193, 0.64)	2.278 (1.428, 12.67)	-0.033 (-0.752, 1.145)	4.860 (1.709, 12.136)	1.431 (0.418, 14.248)
1:5	20.65 (14.22, 24.05)	29.27 (9.92, 69.67)	0.416 (-0.129, 0.73)	3.742 (2.024, 12.83)	-0.084 (-0.711, 1.192)	4.410 (2.954, 12.101)	3.637 (1.220, 19.992)
2:5	22.94 (14.62, 23.46)	43.95 (17.20, 71.80)	-0.004 (-0.455, 0.18)	7.588 (3.105, 21.60)	-0.487 (-1.053, 1.190)	3.929 (2.401, 12.320)	1.137 (0.950, 51.268)
1:1	23.56 (14.84, 23.95)	36.04 (12.56, 68.87)	0.365 (0.127, 0.86)	5.559 (2.235, 18.26)	-0.021 (-0.734, 1.649)	5.000 (3.216, 16.851)	4.973 (1.156, 25.002)
5:2	23.44 (14.47, 23.96)	28.53 (9.98, 66.62)	0.349 (0.140, 0.83)	5.185 (2.700, 21.77)	0.213 (-0.675, 1.135)	4.114 (3.162, 9.824)	4.123 (1.061, 36.873)
5:1	23.11 (14.69, 23.87)	30.54 (11.73, 72.37)	0.331 (0.080, 0.71)	5.279 (2.244, 21.49)	0.288 (-0.574, 1.263)	3.996 (2.772, 21.295)	4.983 (0.994, 22.772)
1:0	22.98 (14.37, 23.76)	25.50 (9.17, 62.71)	0.316 (0.059, 0.53)	6.666 (2.322, 18.31)	0.274 (-0.704, 2.777)	3.937 (2.478, 22.769)	5.786 (0.751, 30.193)

Table 5.3: Insertion/Deletion and Substitution Proportion Fitness
Fitness results of tested insertion/deletion:substitution mutation proportions. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses.

Proportion	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
0:1	0.772 (0.550, 0.993)	0.167 (0.064, 0.743)	0.158 (0.074, 0.162)	0.125 (0.093, 0.649)	1.85×10^{-4} (0.86, 6.03)	0.938 (0.627, 0.988)	4.088×10^{-3} (2.048, 26.287)
1:5	0.874 (0.721, 0.988)	0.388 (0.140, 0.869)	0.156 (0.078, 0.186)	0.209 (0.121, 0.632)	1.84×10^{-4} (0.86, 6.53)	0.954 (0.638, 0.978)	8.088×10^{-3} (4.027, 40.032)
2:5	0.869 (0.705, 0.918)	0.518 (0.222, 0.805)	0.115 (0.063, 0.134)	0.425 (0.168, 0.915)	0.93×10^{-4} (0.00, 6.96)	0.817 (0.732, 0.918)	3.838×10^{-3} (3.746, 94.071)
1:1	0.949 (0.754, 0.978)	0.450 (0.172, 0.852)	0.153 (0.146, 0.231)	0.298 (0.139, 0.944)	2.24×10^{-4} (0.90, 10.33)	0.944 (0.810, 0.967)	10.271×10^{-3} (3.988, 49.302)
5:2	0.944 (0.735, 0.969)	0.379 (0.139, 0.837)	0.151 (0.143, 0.228)	0.310 (0.151, 0.971)	2.74×10^{-4} (0.94, 6.39)	0.926 (0.822, 0.955)	8.741×10^{-3} (3.938, 70.421)
5:1	0.936 (0.735, 0.961)	0.387 (0.162, 0.939)	0.152 (0.121, 0.218)	0.289 (0.138, 0.963)	2.83×10^{-4} (1.02, 7.06)	0.924 (0.821, 0.958)	10.345×10^{-3} (3.910, 42.835)
1:0	0.931 (0.733, 0.963)	0.346 (0.131, 0.801)	0.150 (0.119, 0.176)	0.353 (0.148, 0.970)	2.90×10^{-4} (0.67, 30.43)	0.917 (0.730, 0.952)	11.870×10^{-3} (3.889, 44.913)

Table 5.4: Insertion/Deletion and Substitution Proportion Task Success
Normalized task success results of tested insertion/deletion:substitution mutation proportions. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses.

5.2.3 Per-Site Mutation Rates vs. Per-Genome Mutation Rates

Given a constant genome length, mutation rate denotes a consistent probability distribution that a certain number of mutations would occur. However, when genome length is allowed to vary, the translation of mutation rate to the probability of a certain number of mutations becomes open to multiple potential implementations. Mutations may be determined and applied as a fixed genomic probability, regardless of length variation, which would cause the per-site mutation rate to decrease as genome length increases. Alternatively, the mutation rate can be applied as a per-site/instruction probability of mutation, in which case the genomic level mutation rate will increase as genome size increases.

Similar to mutation timing, described above, the default mutations utilized in Avida use a combination of these mutation rate methods, again originally chosen in a semi-arbitrary manner. By default, instruction substitution mutations are applied as a per-instruction copied rate. Insertion and deletion mutations are a fixed probability of a single instruction insertion or deletion per genome.

I have tested and compared variation in evolutionary potential when utilizing multiple combinations of absolute and per-site relative mutation probabilities. The experiments consisted of six treatment groups and one control group. All groups applied all mutations during offspring division. The overall mutation rate was initially fixed, such that per-site and absolute rates started out identical given the length of the seed genotype.

The first four groups varied the rates of substitution mutations and insertion/deletion mutations as two parameters. The control group utilized all per-site mutation rates. The first treatment group closely replicated Avida defaults, with a per-site substitution rate and absolute probability of single instruction insertion or deletion. The second treatment group

tested the inverse of the Avida defaults, applying insertion/deletion mutations as per-site rates and substitution mutations as an absolute probability of a single mutation. The third treatment group tested both mutation classes as absolute probabilities of single mutation.

In the three treatment groups described so far, all absolute mutation rates were limited to a single occurrence. The per-site mutation rates, in contrast, allowed for multiple mutations to occur in a given generation. I repeated the three previously described mutation treatments, testing Avida defaults, inverted defaults, and all absolute rates, using mutations that were applied with a Poisson process.

Analysis of all seven mutation rate application methods (see Tables 5.5 and 5.6) showed that evolution is generally quite robust to such variation. No significant difference in evolutionary potential was observed in the Logic-9, Match-12, Fibonacci-32, Sort-10, or Navigation environments among any of the tested mutation rate application methods.

The Logic-77 did demonstrate significantly reduced fitness when per-genome rates were used, both single mutation and poisson rate. Task success was also significantly depressed with single mutation per-genome rates. Due to the complexity of the environment, with seventy-seven tasks, it is likely that longer genomes are important for exploiting them. With per-genome rates, the effective per-site mutation rate decreases as the genome length increases. As observed in Section 5.2.1, as the per-site mutation rate decreases the Logic-77 environment demonstrates reduced evolutionary potential, explaining the observations here.

In contrast to the Logic-77 environment, the Limited-9 environment demonstrated significantly increased median fitness and task success with both per-genome calculation methods. This also matches well with the task success results of Section 5.2.1, where the Limited-9 environment shows increased potential with lower per-site mutation rates.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
Per-Site Both	23.09 (14.66, 23.94)	29.17 (9.89, 63.39)	0.368 (0.195, 0.53)	5.099 (2.593, 21.29)	0.006 (-0.745, 1.208)	4.272 (2.735, 8.799)	2.307 (1.123, 21.519)
Per-Site Substitution	23.37 (14.87, 23.96)	30.48 (10.49, 69.02)	0.381 (-0.073, 0.84)	5.693 (2.417, 17.68)	0.072 (-0.813, 1.228)	4.592 (3.414, 19.856)	3.522 (1.104, 21.806)
Per-Genome Substitution	23.51 (14.21, 23.96)	31.06 (10.58, 64.46)	0.366 (0.080, 0.67)	5.591 (1.978, 21.82)	-0.026 (-0.727, 1.835)	4.179 (3.101, 22.910)	4.568 (1.135, 43.903)
Per-Genome Both	19.91 (9.94, 24.25)	**17.90** (6.83, 56.20)	0.362 (0.037, 0.54)	3.802 (1.470, 13.12)	0.226 (-0.666, 1.281)	**6.465** (3.722, 23.996)	3.531 (0.975, 18.637)
Poisson Indel	23.47 (14.88, 24.02)	29.50 (9.95, 64.11)	0.373 (-0.108, 0.82)	5.207 (2.509, 13.21)	-0.048 (-0.702, 0.939)	4.425 (3.274, 18.366)	4.380 (0.990, 25.160)
Poisson Substitution	23.55 (14.83, 24.02)	35.40 (13.43, 66.69)	0.391 (0.200, 0.83)	5.475 (1.962, 13.12)	0.124 (-0.695, 1.154)	4.478 (3.416, 13.833)	2.601 (0.906, 17.380)
Poisson Both	22.10 (11.83, 24.23)	**18.81** (4.44, 68.94)	0.367 (0.132, 0.81)	3.759 (1.481, 13.06)	0.168 (-0.752, 1.716)	**4.963** (3.493, 23.831)	3.635 (0.986, 12.653)

Table 5.5: Mutation Rate Application Fitness

Fitness results of tested mutation rate applications methods. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
Per-Site Both	0.948 (0.744, 0.971)	0.390 (0.138, 0.785)	0.153 (0.145, 0.158)	0.289 (0.149, 0.908)	2.25×10^{-4} (0.90, 6.22)	0.945 (0.809, 0.960)	5.646×10^{-3} (3.958, 39.625)
Per-Site Substitution	0.950 (0.746, 0.978)	0.414 (0.146, 0.885)	0.153 (0.124, 0.227)	0.311 (0.139, 0.975)	2.35×10^{-4} (0.00, 6.84)	0.942 (0.826, 0.966)	8.041×10^{-3} (3.694, 40.570)
Per-Genome Substitution	0.953 (0.709, 0.975)	0.420 (0.146, 0.822)	0.154 (0.144, 0.230)	0.304 (0.122, 0.931)	2.09×10^{-4} (0.54, 11.44)	0.949 (0.781, 0.978)	8.995×10^{-3} (3.828, 76.636)
Per-Genome Both	0.874 (0.556, 0.994)	**0.247** (0.102, 0.701)	0.153 (0.147, 0.162)	0.207 (0.112, 0.622)	1.92×10^{-4} (0.92, 6.52)	**0.951** (0.692, 0.995)	8.137×10^{-3} (3.914, 35.491)
Poisson Indel	0.949 (0.755, 0.977)	0.396 (0.139, 0.779)	0.153 (0.117, 0.230)	0.278 (0.149, 0.628)	2.06×10^{-4} (0.62, 4.78)	0.946 (0.911, 0.966)	9.680×10^{-3} (3.528, 47.875)
Poisson Subsitution	0.958 (0.754, 0.979)	0.450 (0.183, 0.859)	0.153 (0.143, 0.228)	0.289 (0.121, 0.614)	2.25×10^{-4} (0.59, 5.54)	0.950 (0.896, 0.974)	6.152×10^{-3} (3.644, 34.698)
Poisson Both	0.916 (0.606, 0.992)	0.260 (0.073, 0.927)	0.153 (0.149, 0.230)	0.208 (0.107, 0.630)	2.17×10^{-4} (0.88, 8.59)	**0.954** (0.724, 0.989)	8.176×10^{-3} (3.918, 24.294)

Table 5.6: Mutation Rate Application Task Success

Normalized task success results of tested mutation rate application methods. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

5.3 Investigating the Effect of Mutation Timing

Genotypic mutations in natural systems may arise during the process of copying genetic material. In digital evolution systems, mutations can be modeled to produce similar behavior, producing mutations during the actual execution of the copy operation. However, unlike natural systems, it is also possible to limit the timing of all mutations such that they occur only during division of offspring, to toggle whether mutations are also applied to the parent genome, to trigger mutations randomly at any time (cosmic-ray mutations), among other possibilities, providing the experimenter with greater control of the exact characteristics of the mutation.

5.3.1 Copy Mutations versus Divide Mutations

There are many potential side effects associated with the timing of mutation application. Mutations that occur during the copy process can affect the execution of the current organism, whereas mutations on divide cannot. Intra-lifetime copy mutations could possibly allow the evolution of error correction, provided that the genetic language supports it. Mutations applied on divide can be exactly calibrated to genome length or applied with absolute probability, whereas copy mutations may only be produced reliably as per-site rates.

In Avida, the default mutations use a combination of copy process mutations and mutations on divide, chosen for experiment run-time efficiency. The consequences of this combination, and the effect on evolutionary potential of choosing other combinations have not been explored. I have tested this with four treatments varying the timing of mutations. The control treatment applied all mutations on-divide, as described in Chapter 2 and used in all other studies. One treatment used mutations similar to the Avida defaults, with copy

process substitutions and on-divide insertion and deletion mutations.[1] The inverted treatment applied insertion and deletion mutations during the copy process, while instruction substitutions occurred on-divide. The final treatment group applied all mutations during the copy process, with no mutations on divide.

The results, presented in Tables 5.7 and 5.8, showed that the Avida default treatment, with on-copy substitution mutations and on-divide insertion and deletion mutations, and the on-divide treatment both demonstrated the greatest evolutionary potential. The inverted and all on-copy treatments, featuring copy process insertion and deletion mutations, demonstrated significantly reduced task success and fitness in many environments, indicating that such mutations are likely disruptive and present a drag on evolutionary potential.

[1] Avida default mutation rates apply a probability of instruction substitution per instruction copied during the organism's lifetime and an absolute probability each of single-instruction insertion or deletion on division of the offspring. In order to maintain consistent genomic mutation rates, insertion and deletion mutations were applied using per-site rates compensated for the ancestral genotype length.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
On-Divide	23.64 (14.77, 24.00)	32.98 (11.53, 68.36)	0.377 (0.179, 0.85)	5.507 (2.629, 22.04)	0.224 (-0.693, 2.159)	4.278 (2.638, 10.425)	4.055 (1.075, 62.130)
Default	23.44 (14.87, 23.96)	35.19 (9.62, 70.67)	0.367 (0.120, 0.80)	5.447 (1.937, 21.94)	0.020 (-0.713, 1.225)	4.187 (2.984, 7.240)	4.371 (1.049, 63.064)
Inverted	**22.74** (14.00, 23.79)	41.42 (10.77, 72.54)	0.363 (0.015, 0.64)	7.079 (2.358, 21.58)	**-0.266** (-1.073, 0.884)	4.495 (2.912, 20.787)	**1.096** (0.822, 21.592)
On-Copy	**22.83** (14.20, 23.69)	39.42 (7.81, 71.19)	**0.330** (-0.022, 0.71)	7.740 (2.188, 21.72)	**-0.060** (-1.185, 1.053)	3.922 (3.152, 19.155)	2.184 (0.592, 107.041)

Table 5.7: Mutation Timing Fitness

Fitness results of tested mutation timings. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
On-Divide	0.943 (0.744, 0.971)	0.403 (0.161, 0.799)	0.153 (0.140, 0.225)	0.299 (0.149, 0.946)	2.59×10^{-4} (0.94, 21.57)	0.941 (0.801, 0.975)	8.661×10^{-3} (3.964, 117.995)
Default	0.951 (0.743, 0.975)	0.465 (0.134, 0.877)	0.153 (0.147, 0.222)	0.289 (0.121, 0.954)	2.44×10^{-4} (0.58, 6.08)	0.944 (0.831, 0.962)	9.150×10^{-3} (3.964, 100.548)
Inverted	0.934 (0.730, 0.966)	0.548 (0.149, 0.926)	**0.149** (0.141, 0.216)	0.398 (0.148, 0.958)	$\mathbf{1.96\times10^{-4}}$ (0.89, 7.00)	**0.917** (0.767, 0.953)	$\mathbf{3.999\times10^{-3}}$ (3.913, 41.426)
On-Copy	0.933 (0.736, 0.966)	0.528 (0.118, 0.887)	**0.148** (0.141, 0.216)	0.446 (0.148, 0.962)	2.33×10^{-4} (0.87, 7.19)	**0.919** (0.796, 0.967)	5.007×10^{-3} (3.673, 200.922)

Table 5.8: Mutation Timing Task Success

Normalized task success results of tested mutation timings. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

5.3.2 Cosmic-Ray Mutations

Copy mutations represent variation introduced during the replication process, internally created and typically limited in scope to future offspring.[2] Externally induced mutations, sometimes referred to as "cosmic-ray" mutations, however, are not limited in scope. Such mutations introduce broad variation into the population, potentially affecting both current and future generations.

I have tested the effects of cosmic-ray mutations on evolutionary potential relative to the control mutation settings. Canonical Avida cosmic-ray mutations perform single instruction substitution at a given fixed rate. In order to fairly compare with the control, which features both substitution and insertion/deletion mutations, I implemented and tested cosmic-ray insertion/deletion mutations alongside instruction substitution cosmic-ray mutations. Since cosmic-ray mutations are applied as a function of time, rather than genome length, I set the mutation rate to be approximately 0.3 genomic by factoring in the initial gestation time of the ancestral organism.

In all seven environments, cosmic-ray mutations demonstrated decreased median task success (Table 5.10). Among these, the Fibonacci-32 and Limited-9 environments were significant with 40.4% and 54.6% drops, respectively. Fitness results were similarly depressed six of the seven environments, with the Fibonacci-32, Limited-9, and Navigation significantly so (Table 5.9). The Logic-77 environment was the only environment to feature increased median fitness, though the results were statistically identical to the control. Overall these results indicate that cosmic-ray mutations, which may disrupt an organism's genome during its lifetime, are detrimental to the evolutionary potential of the system.

[2]It is technically possible for intra-lifetime copy mutations to overwrite portions of the parental genome.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
Control	23.23 (14.59, 23.96)	36.05 (11.56, 65.00)	0.375 (0.075, 0.72)	5.673 (2.665, 21.99)	-0.011 (-0.734, 1.342)	3.993 (2.681, 9.328)	4.221 (1.141, 34.603)
Cosmic-Ray	22.86 (12.68, 23.84)	37.95 (8.44, 65.99)	0.361 (0.220, 0.75)	**4.651** (2.279, 21.24)	-0.189 (-1.027, 1.345)	**3.226** (1.426, 6.248)	**1.202** (1.003, 19.167)

Table 5.9: Cosmic-Ray Mutation Fitness

Fitness results of cosmic-ray mutations. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
Control	0.945 (0.736, 0.970)	0.450 (0.161, 0.823)	0.153 (0.116, 0.224)	0.305 (0.151, 0.961)	2.19×10^{-4} (0.92, 6.72)	0.942 (0.792, 0.968)	8.961×10^{-3} (3.964, 67.347)
Cosmic-Ray	0.939 (0.646, 0.970)	0.426 (0.126, 0.721)	0.153 (0.147, 0.224)	**0.182** (0.127, 0.870)	1.95×10^{-4} (0.00, 8.06)	**0.934** (0.560, 0.954)	4.069×10^{-3} (4.021, 35.548)

Table 5.10: Cosmic-Ray Mutation Task Success

Normalized task success results of cosmic-ray mutations. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

5.3.3 Parent Mutations

During the process of division in biological organisms the resulting organisms may represent a distinct parent and offspring pair. Alternatively, the resulting organisms may be more correctly treated as two daughter cells, with genetic material and cellular machinery somewhat equally distributed among the offspring. During the formation of daughter cells, mutations accumulated during the replication process may end up in either of the offspring. As a result of this, even though genomic mutation rate may remain constant, the generational mutation rate could be substantially larger.

I have explored the evolutionary potential consequences of this difference. I compared two experimental groups featuring parental mutations applied on divide relative to the standard control configuration. The first experiment group featured parent-organism mutations identical in class and probability to those of offspring. The net effect of this change is that generational mutation rate was approximately double that of the control group. In the second experiment group, I tested parent and offspring mutation rates that were half that of the first group. This change yielded a generational mutation rate comparable to the control group, but featured a corresponding decrease in individual organism mutation rate.

As a mutation timing, parent mutations made no significant difference in evolutionary potential (see Tables 5.11 and 5.12). The Logic-77, Fibonacci-32, and Sort-10 environments were statistically equivalent under both treatments. The Logic-9, Match-12, Limited-9 and Navigation environments demonstrated significant reductions in task success and often fitness in the standard parent mutations treatment. However, when compensated for the generational mutation rate in the parent half-rate treatment, these differences disappeared.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
CONTROL	22.36 (14.47, 23.92)	28.30 (8.91, 70.21)	0.375 (0.029, 0.72)	5.192 (2.025, 21.72)	0.250 (-0.713, 1.731)	4.343 (3.124, 10.958)	4.256 (1.104, 21.705)
PARENT	22.63 (13.53, 23.79)	31.33 (10.41, 66.55)	**0.204** (0.029, 0.40)	5.502 (2.634, 21.36)	-0.081 (-0.747, 3.574)	4.019 (2.808, 8.415)	**1.267** (1.083, 59.015)
PARENT HALF-RATE	22.29 (13.03, 23.93)	31.55 (7.63, 61.80)	0.369 (-0.004, 0.68)	5.043 (2.375, 20.64)	-0.045 (-0.768, 1.212)	4.009 (2.521, 7.815)	3.602 (1.077, 22.752)

Table 5.11: Parent Mutation Fitness

Fitness results of parent mutations and parent mutations at half-rate. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
CONTROL	0.894 (0.743, 0.969)	0.362 (0.127, 0.818)	0.153 (0.120, 0.223)	0.293 (0.121, 0.962)	2.68×10^{-4} (0.00, 9.63)	0.943 (0.850, 0.962)	8.681×10^{-3} (3.978, 40.146)
PARENT	**0.890** (0.685, 0.956)	0.390 (0.146, 0.787)	**0.140** (0.123, 0.149)	0.277 (0.147, 0.886)	2.00×10^{-4} (0.00, 41.22)	**0.895** (0.833, 0.938)	$\mathbf{3.980\times10^{-3}}$ (3.938, 109.987)
PARENT HALF-RATE	0.898 (0.646, 0.976)	0.423 (0.110, 0.781)	0.152 (0.142, 0.229)	0.292 (0.139, 0.974)	2.11×10^{-4} (0.80, 6.48)	0.939 (0.746, 0.958)	8.337×10^{-3} (3.993, 44.897)

Table 5.12: Parent Mutation Task Success

Normalized task success results of parent mutations and parent mutations at half-rate. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

5.4 Exploring Aspects of Insertion/Deletion Mutations of Large Effect

Mutations commonly used in digital evolution systems typically affect only one instruction at a time. These mutations can accumulate over generations, producing complex features (Lenski *et al.*, 2003). Natural systems, however, are not limited to single-site mutations. Mutations of large-effect can occur, producing duplicate genes, chromosomes, or even whole genomes, and play an important role in biological evolution (Lynch, 2000; Zhang, 2003).

The consequences of multi-site mutations can vary widely. Given the potential size and scope of the genome changes, they may result in significant jumps through genotype space. The creation of duplicate genetic material, including potentially complete genes, may allow for the evolution of novel or improved functionality that would otherwise have been unlikely. One copy can be altered or used as a building block for a new trait while the organism retains the original functionality using the other copy.

I systematically tested versions of an insertion/deletion mutation of large effect in Avida, called a "slip" mutation. Slip mutations produce genetic variation that simulates the genome replication mechanism shifting while reading the source genome, causing wholesale duplication or deletion of contiguous genetic segments. Slip mutations, as implemented in Avida, support four versions of the duplication (insertion) behavior[3], as summarized in Table 5.13. Deletion behavior is identical for all versions. The size of resulting insertions and deletions is limited to the length of the genome, selected as occurring at a random upstream or

[3]Slip mutations in Avida were originally conceived and implemented by Jeff Barrick. I have extended both the mutations themselves and the analysis of their effect on evolutionary potential.

Name	Description of Inserted Code
duplicate	An identical copy of the previous region of code
scramble	A randomly reordered copy of the previous region of code
random	A completely random sequence of instructions
nop-X	A series of nop-X instruction (that are as neutral as possible)

Table 5.13: Insertion versions supported by the Slip mutation in Avida

downstream location with uniform probability.

Results from a preliminary study, conducted in the Logic-9 environment, indicated that slip mutation versions that inserted duplicate or scrambled genomic program segments significantly improved evolutionary potential. The increased potential was borne out both as a more rapid acquisition of fitness relative to all other treatments, and as a greater median final fitness. Building off of this preliminary work, I have systematically explored how slip mutations impact evolutionary potential across all seven environments.

5.4.1 Mutation Rates

Mutation rate and the balance of substitution and insertion/deletion mutations both play critical roles in determining evolutionary potential, as demonstrated previously in this chapter. The rate at which slip mutations occur likely needs to be similarly calibrated for optimal evolutionary potential with respect to the standard mutations. I have examined the evolutionary potential of slip mutations, featuring duplication insertion sequences, at four distinct rates proportional to the substitution and insertion/deletion rates, 0.3, 0.1, 0.03, and 0.01, while holding the overall mutation rate of the system constant at 0.3 genomic.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
Control	23.37 (14.10, 23.94)	28.67 (9.12, 63.77)	0.361 (0.181, 0.87)	4.691 (1.643, 8.56)	0.028 (-0.650, 1.238)	4.395 (3.098, 13.742)	6.275 (1.089, 21.199)
0.3	**23.98** (-0.23, 24.33)	**44.84** (-0.23, 71.46)	0.288 (-0.008, 0.85)	**7.018** (3.506, 8.08)	**-0.223** (-0.239, -0.210)	**6.310** (-0.227, 14.118)	**1.828** (0.817, 17.660)
0.1	**23.87** (-0.23, 24.22)	**42.73** (-0.23, 72.00)	**0.218** (-0.041, 0.74)	**7.373** (3.256, 21.52)	**-0.231** (-0.284, 0.037)	**7.584** (-0.239, 23.278)	1.828 (0.807, 30.084)
0.03	**23.91** (15.07, 24.19)	**37.87** (21.17, 67.94)	0.295 (-0.237, 0.71)	**5.518** (3.250, 12.89)	**-0.239** (-0.297, 0.615)	**7.798** (-0.152, 23.476)	4.151 (0.814, 28.909)
0.01	**23.90** (8.68, 24.10)	**38.69** (21.01, 70.97)	0.317 (-0.232, 0.74)	**6.780** (3.124, 20.53)	**-0.272** (-0.696, 0.433)	**6.564** (2.298, 23.479)	3.750 (1.456, 23.998)

Table 5.14: Slip Duplication Mutation Rate Fitness

Fitness results for tested slip duplication mutation rates. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
Control	0.950 (0.712, 0.976)	0.377 (0.130, 0.794)	0.153 (0.142, 0.227)	0.239 (0.112, 0.463)	2.29×10^{-4} (0.000, 0.001)	0.942 (0.844, 0.961)	13.39×10^{-3} (0.004, 0.035)
0.3	0.956 (0.000, 0.978)	0.506 (0.000, 0.788)	0.154 (0.078, 0.212)	**0.347** (0.183, 0.472)	**0.0** (0.000, 0.000)	0.951 (0.000, 0.973)	$\mathbf{4.05\times10^{-3}}$ (0.002, 0.034)
0.1	0.957 (0.000, 0.977)	0.477 (0.000, 0.877)	0.154 (0.100, 0.229)	**0.358** (0.171, 0.927)	**0.0** (0.000, 0.000)	0.940 (0.000, 0.969)	4.07×10^{-3} (0.002, 0.055)
0.03	**0.964** (0.582, 0.977)	0.423 (0.269, 0.790)	0.155 (0.000, 0.222)	0.282 (0.171, 0.629)	**0.0** (0.000, 0.000)	0.944 (0.069, 0.964)	8.25×10^{-3} (0.002, 0.052)
0.01	**0.963** (0.427, 0.980)	0.431 (0.277, 0.891)	**0.155** (0.000, 0.228)	0.319 (0.159, 0.943)	**0.000** (0.000, 0.000)	0.944 (0.598, 0.963)	8.16×10^{-3} (0.004, 0.041)

Table 5.15: Slip Duplication Mutation Rate Task Success

Normalized task success results for tested slip duplication mutation rates. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

Fitness results, presented in Table 5.14, show that slip mutations at all tested rates significantly increase evolutionary potential in the Logic-9, Logic-77, Fibonacci-32, and Limited-9 environments. The Match-12 and Navigation environments demonstrated the opposite, with reduced median fitness under all rates, though generally statistically identical to the control. The Sort-10 environment demonstrates substantial loss of fitness with all tested rates.

Task success results, shown in Table 5.15, demonstrate shifts that correlate with fitness. However, unlike the fitness results, most of the observed differences are not significant. Only Logic-9, with the 0.03 and 0.01 rates, and Fibonacci-32, with the 0.3 and 0.1 rates, demonstrate notable improvements.

These results indicate that slip mutations may present a tradeoff in determining evolutionary potential. Some environments yield improved results, while others experience the opposite. However, the inflection point in evolutionary potential is largely insensitive to the range of rates that I have tested here. Given that, I arbitrarily selected the 0.1 rate for use in all of the remaining experiments in this section.

5.4.2 Insertion Sequence Content

Slip mutations have demonstrated improved evolutionary potential in a subset of the environments explored here, the Logic-9, Logic-77, Fibonacci-32, and Limited-9 environments. There are multiple possible explanations for why slip mutations help those environments. Gene (or subroutine) duplication of complete functionality may be essential for creating and exploiting building blocks of functionality. Alternatively, it may simply be large changes in genome length that is beneficial, inserting chunks of unused genetic code. Existing somewhere between these two extremes of duplicated code and random (or blank) genetic material,

scrambled genetic sequences may allow for favorable distributions of instructions while disentangling epistasis present in previously evolved functions. In order to test each of these explanations, I implemented four versions of the slip mutation that vary the content of the inserted sequence (see Table 5.13). Each of these versions was tested in all seven environments.

The Slip-Random and Slip-Nop-X versions, testing two forms of "blank" tape, were both statistically equivalent to the control in all but the Sort-10 and Navigation environments for both fitness and task success (see Tables 5.16 and 5.17). The Sort-10 environment demonstrated significantly reduced evolutionary potential with both insertion sequence versions, while the Navigation environment was affected negatively only by the Slip-Nop-X version.

The Slip-Duplicate and Slip-Scramble versions both demonstrated equivalent evolutionary potential, substantially improving fitness results in the Logic-9, Logic-77, and Limited-9 environments. The Fibonacci-32 environment showed increased median fitness, though these were not significant after Bonferonni correction. The Match-12, Sort-10, and Navigation environments all had reduced performance with these two slip mutation versions.

Overall, these results match well with previous observations, indicating that environments helped by slip mutations benefit equally from duplicated and scrambled insertion sequences. The distribution of instructions in inserted sequences contributes substantially to their utility for exploiting evolutionary potential. However, the utility of slip mutations remains sensitive to the environment, regardless of insertion content.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
CONTROL	21.58 (12.86, 24.00)	30.66 (8.57, 60.50)	0.372 (0.140, 0.67)	5.132 (1.932, 22.22)	0.109 (-0.740, 1.324)	4.298 (2.837, 14.264)	4.002 (1.111, 25.834)
SLIP-DUPLICATE	**23.95** (-0.24, 24.26)	**43.77** (-0.23, 71.54)	**0.241** (-0.001, 0.69)	5.969 (3.898, 14.79)	-0.226 (-0.267, -0.051)	**8.480** (2.253, 22.833)	1.821 (0.776, 25.595)
SLIP-SCRAMBLE	**24.03** (-0.23, 24.23)	**44.50** (-0.23, 65.88)	**0.276** (0.044, 0.72)	5.925 (3.706, 8.33)	-0.229 (-0.284, 0.576)	**6.539** (2.088, 17.902)	**1.809** (0.821, 5.426)
SLIP-RANDOM	23.14 (-0.35, 23.99)	29.01 (7.06, 66.41)	0.301 (-0.376, 0.66)	5.849 (1.696, 8.31)	-0.352 (-0.522, -0.316)	**3.546** (0.497, 11.188)	3.655 (1.325, 29.682)
SLIP-NOP-X	22.38 (-0.38, 24.02)	31.63 (-0.39, 68.59)	0.309 (-0.047, 0.72)	4.769 (1.367, 22.10)	-0.362 (-0.466, -0.107)	**3.792** (-0.387, 14.290)	**0.981** (0.555, 11.855)

Table 5.16: Slip Mutation Insertion Versions Fitness

Fitness results of all four slip mutation versions. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
CONTROL	0.910 (0.643, 0.973)	0.391 (0.123, 0.792)	0.153 (0.145, 0.223)	0.291 (0.121, 0.963)	2.42×10^{-4} (0.90, 7.08)	0.941 (0.792, 0.969)	8.482×10^{-3} (3.964, 51.545)
SLIP-DUPLICATE	**0.963** (0.000, 0.977)	0.503 (0.000, 0.872)	0.153 (0.079, 0.228)	0.294 (0.199, 0.625)	$\mathbf{0.00\times10^{-4}}$ (0.00, 0.98)	0.943 (0.599, 0.970)	4.063×10^{-3} (2.038, 46.883)
SLIP-SCRAMBLE	**0.965** (0.000, 0.979)	0.492 (0.000, 0.805)	0.154 (0.119, 0.225)	0.291 (0.180, 0.452)	$\mathbf{0.00\times10^{-4}}$ (0.00, 3.73)	0.950 (0.572, 0.966)	$\mathbf{4.041\times10^{-3}}$ (2.040, 10.204)
SLIP-RANDOM	0.949 (0.000, 0.976)	0.391 (0.100, 0.815)	0.154 (0.000, 0.223)	0.316 (0.093, 0.453)	$\mathbf{0.00\times10^{-4}}$ (0.00, 1.05)	0.946 (0.483, 0.968)	8.214×10^{-3} (3.563, 56.061)
SLIP-NOP-X	0.869 (0.000, 0.972)	0.389 (0.000, 0.823)	0.153 (0.125, 0.229)	0.264 (0.081, 0.970)	$\mathbf{0.00\times10^{-4}}$ (0.00, 1.89)	0.946 (0.000, 0.964)	$\mathbf{2.815\times10^{-3}}$ (2.030, 23.641)

Table 5.17: Slip Mutation Insertion Versions Task Success

Normalized task success results of all four slip mutation versions. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
Control	23.03 (14.80, 24.02)	30.73 (10.93, 62.77)	0.378 (0.032, 0.50)	4.817 (2.350, 8.31)	0.018 (-0.726, 1.325)	4.142 (3.134, 10.092)	1.316 (1.087, 14.637)
Trans-Duplicate	**23.89** (9.38, 24.15)	**42.62** (-0.23, 64.89)	**0.250** (0.047, 0.61)	**7.068** (4.052, 13.23)	**-0.230** (-0.702, 0.740)	**6.961** (4.132, 14.846)	**4.186** (0.808, 33.287)
Trans-Scramble	**23.91** (9.65, 24.10)	**43.95** (23.49, 70.03)	**0.272** (-0.093, 0.62)	**6.064** (3.222, 13.04)	**-0.230** (-0.239, 0.398)	**6.391** (2.464, 9.906)	1.816 (0.817, 16.947)

Table 5.18: Gene Translocation Fitness

Fitness results of both gene translocations. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
Control	0.943 (0.749, 0.975)	0.403 (0.151, 0.809)	0.152 (0.117, 0.159)	0.237 (0.138, 0.458)	2.25×10^{-4} (0.91, 6.94)	0.942 (0.803, 0.965)	4.037×10^{-3} (3.967, 26.673)
Trans-Duplicate	**0.965** (0.472, 0.977)	0.491 (0.000, 0.760)	0.154 (0.129, 0.228)	**0.343** (0.210, 0.644)	$\mathbf{0.00\times10^{-4}}$ (0.00, 3.50)	**0.949** (0.922, 0.964)	8.398×10^{-3} (2.042, 64.116)
Trans-Scramble	**0.963** (0.486, 0.973)	**0.505** (0.301, 0.811)	0.154 (0.075, 0.219)	**0.294** (0.171, 0.616)	$\mathbf{0.00\times10^{-4}}$ (0.00, 2.28)	**0.949** (0.600, 0.969)	4.048×10^{-3} (2.046, 32.459)

Table 5.19: Gene Translocation Task Success

Normalized task success results of both gene translocation versions. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

5.4.3 Gene Translocation

The development of the slip mutation, as tested above, was inspired by the idea that the mechanisms responsible for replication could "slip" and copy or skip regions of the genome. The controls detailed above allow for several variations in the distribution of instructions that may be inserted into a region, including copied and scrambled from the region immediately surrounding the current copy location. The source of these fragments, however, was restricted to be immediately adjacent to the insertion location. Gene translocation relaxes that restriction, allowing the genome fragment to be selected from any location within the current genome. I tested the impact of this relaxation with two new versions of the slip mutation. These versions support the insertion of copied and scrambled sequences selected at random from any location in the genome.

Gene translocation, in both duplication and scrambled forms, demonstrated significantly improved fitness in the Logic-9, Logic-77, Fibonacci-32, and Limited-9 environments (see Tables 5.18 and 5.19). These results match well with conventional slip mutations, tested above. In contrast to slip mutations, gene translocation mutations demonstrated correspondingly substantial improvements in task success in all but one treatment, Trans-Duplication in the Logic-77 environment. Additionally, the Navigation environment demonstrated substantially improved fitness with Trans-Duplication mutation. The Match-12 and Sort-10 environments, on the other hand, demonstrated significantly reduced evolutionary potential, similar to previous slip mutation results.

Overall, these results indicate that limiting the source of genetic insertions to neighboring regions of the genome, as featured in slip mutations, is unnecessary. Indeed, relaxing that restriction with gene translocation mutations unlocks additional evolutionary potential in

favorable environments and may help mitigate the detrimental effects in others.

5.4.4 Discussion

Slip mutations that insert duplicated or scrambled genetic code substantially increase evolutionary potential, especially in the gene translocation form. Five of the environments, the Logic-9, Logic-77, Fibonacci-32, Limited-9, and Navigation environments all benefited from these mutations. The Match-12 and Sort-10 environments did show reduced fitness and task success with these mutations. However, as observed throughout Chapter 4, both of these environments are particularly difficult. The instruction set, especially the HEADS architecture used in this chapter, simply does not have computational power necessary to effectively tackle the Sort-10 environment. Similarly, the lack of support for building arbitrary numbers likely limits the potential in the Match-12 environment. Given instruction set improvements supporting these two environments, it is possible that slip mutations may positively support the evolutionary potential of populations in them.

5.5 Side-effects of Mutations

Unstable genotypes produce offspring whose genomes differ from the parent, even in the context of no new explicit mutations occurring. These "implicit mutations" result from errors in the replication algorithm of the parent organism. Avida offers the ability to prevent unstable genotypes from propagating in order to allow for full control of the exact mutation characteristics of an experiment. The consequences of this control mechanism on evolutionary potential, however, have not previously been explored. Preventing the reproduction of these genotypes causes what may otherwise be neutral or beneficial mutations to be converted into lethal mutations. I have explored the ramifications of sterilizing unstable genotypes within my evolutionary potential framework.

In general, sterilizing unstable genotypes made no substantial impact on evolutionary potential in all but one environment, the Logic-77 environment (see Tables 5.20 and 5.21). Although the Match-12, Limited-9, and Navigation environments each featured a significant variation in either fitness or task success, none were accompanied with a corresponding change in the other metric. The Logic-77 environment, however, demonstrated highly significant and substantial declines in both median $\log_2$ fitness (47.8%) and median normalized task success (44.3%), indicating that in this environment unstable genotypes play an important role in exploring the fitness landscape to exploit evolutionary potential.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
CONTROL	23.19 (13.06, 23.91)	37.11 (9.90, 66.25)	0.352 (-0.053, 0.64)	4.955 (1.894, 13.24)	0.162 (-0.733, 1.164)	4.317 (3.415, 19.058)	4.125 (1.126, 25.009)
STERILIZE UNSTABLE	23.00 (14.06, 23.95)	**19.38** (8.86, 56.23)	**0.399** (0.218, 0.78)	5.344 (1.858, 22.01)	-0.020 (-0.693, 1.288)	4.192 (3.318, 9.398)	**2.720** (1.035, 40.186)

Table 5.20: Sterilize Unstable — Fitness

Fitness results when sterilizing unstable genotypes. Each entry shows the median $\log_2$ population mean fitness in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

	Logic-9	Logic-77	Match-12	Fibonacci-32	Sort-10	Limited-9	Navigation
CONTROL	0.950 (0.648, 0.978)	0.470 (0.139, 0.847)	0.153 (0.077, 0.182)	0.274 (0.122, 0.638)	2.54×10^{-4} (0.56, 5.55)	0.942 (0.802, 0.966)	8.500×10^{-3} (3.965, 43.938)
STERILIZE UNSTABLE	0.960 (0.694, 0.976)	**0.262** (0.126, 0.717)	0.153 (0.148, 0.223)	0.291 (0.122, 0.992)	2.13×10^{-4} (0.95, 6.96)	**0.954** (0.786, 0.970)	6.210×10^{-3} (3.813, 79.650)

Table 5.21: Sterilize Unstable Genotypes — Task Success

Normalized task success results when sterilizing unstable genotypes. Each entry shows the median normalized task success in the respective environment, with 95% confidence intervals in parentheses. Bold entries indicate significant ($p < 0.05$) deviations.

5.6 Conclusion

Exploration of four aspects of mutations (types of mutations, timing of mutation application, mutations of large effect, and the side-effects of mutations) highlighted that digital evolution is **surprisingly robust to variation in mutations**. Although some minor variation in specific environment and mutation combinations was observed, the overall mutation rate demonstrated the greatest impact on evolutionary potential.

5.7 Future Work

There are many potential avenues of further exploration that would be worthwhile investigating, but are beyond the scope of this dissertation (indeed, some could be the topic of an entire dissertation unto themselves). Here I describe several possibilities that I considered.

5.7.1 Comparing Sexual and Asexual Reproduction

Genetic recombination, such as which occurs during sexual reproduction, has been shown to have a broad range of effects on genetic architecture (Misevic *et al.*, 2006). Evolutionary computation makes frequent use of it as a primary search operator, especially in genetic algorithms (Holland, 1975). The effects on evolutionary potential of sexual reproduction could be systematically explored in digital organisms under multiple recombination probabilities, module numbers, genome alignment techniques, and mate-selection models.

5.7.2 Execution Errors

Up to this point, all mutation types discussed create lasting changes to the genome of offspring, parents, or both. Mutations in natural systems are not limited to this level of activity. Indeed, it is possible for errors to occur at almost any level. Execution errors are thought to be a strong force in evolving more robust code, and especially redundancies (that otherwise wouldn't have a strong selective pressure). In digital organisms, however, the virtual CPU activity is generally error free, producing deterministic results. Introducing execution errors and/or failures, however, may have consequences on the structure and robustness of the evolved genetic program.

5.7.3 Mutation Changes During Evolution

All of the work proposed above examines mutations as discrete, fixed sets. In natural evolution, however, the mechanisms of replication and expression are themselves evolving. Through this evolution, new types of mutations may become possible, rates of mutations can change, the distribution of substitutions may vary. Indeed almost all characteristics of mutations can change over time.

It is possible, in principle, for implicit mutations to arise out of the replication process in a digital evolution system, altering the mutations of the system. These specific types of mutations are difficult to intentionally create and control. However, it is possible to explicitly change mutation rates during a run. Utilizing such changes, the consequences of changing mutations during evolution could be systematically explored.

Chapter 6

Conclusion and Future Work

The research that I have conducted here contributes generally to our understanding of how history, mutations, and genetic hardware affect evolutionary potential in evolving systems. The results provide insights applicable for the prediction of future evolutionary directions. Knowledge of the roles these aspects play in determining evolution potential will have important implications in both evolutionary computation and evolutionary biology, allowing for the alteration of potential as desired. My work has shown that evolution is quite robust in the face of all of these factors. Most changes are innocuous, yielding little impact on the overall evolutionary potential of populations. Some changes, however, can be critical. In the history case, for example, *most* mutations will make no difference to the evolutionary outcomes at all, but the right mutation can create a strong shift in potential. Still, multiple instances demonstrated the possibility for significantly improving performance.

The work presented will facilitate improvements in our ability to harness digital evolution. Beyond specific improvements, the work demonstrates generalizable observations, methods, and system design guidelines for maximizing evolutionary potential. Importantly, the work

also demonstrates the robustness of the evolutionary process to perturbations of fundamental aspects. In many ways, this helps us understand how evolution functions so well on Earth, where many of the rules are quite arbitrary.

The work presented in this dissertaion covers three major aspects of evolution that affect evolutionary potential. I did not explore spatial structure, affecting gene flow and growth rates. The interplay between spatial structure and mutations, including sexual reproduction, is known to be important, but many details of how and why remain unclear. As continuation of this work, I plan to pursue the exploration of spatial structure and its effect on evolutionary potential.

In the discussion above I indicated that mutations and genetic hardware both have potential implications for genomic architecture, and that genetic architecture may be responsible for some of the observed contingencies of history. For example, I think high levels of epistasis will reduce evolutionary potential because the landscape becomes too rugged to easily traverse. Examining epistasis might give us many insights into the evolutionary potential of a population. I believe that important insights would be found investigating exactly how the above changes should alter genetic architecture. These insights may inform and improve future genetic hardware changes, as well as mutation design and selection.

GLOSSARY

Glossary

head Hardware element that points to an instruction located in the genomic program.. 6, 7, 41, 54

mutational neighborhood All potential mutant genotypes with up to a specified number of mutations surrounding a reference genotype. 14, 23, 24, 32, 35, 36

nop no-operation instruction. 7–9, 40–43, 46, 48, 77

PLoD principal line of descent. 24

principal line of descent The complete genotypic lineage of the most abundant genotype at the end of a given experiment, starting from the ancestral genotype.. 18, 23, 24, 32, 36, 78

STEP short-term evolutionary potential. 14, 23, 32, 35, 78, 79, 81

APPENDICES

Appendix A

Complete History Mutational Neighborhood Results

The mutational neighborhood results for all lineages analyzed in Section 3.2.2 are presented here. In each figure, the top graph shows the absolute probability of a single mutation conferring performance of the EQU function. The middle graph of each figure shows the absolute probability of two mutations occurring simultaneously that, combined, confer performance of EQU. Lastly, the bottom graph shows the quality of individual mutations within two-step pathways when taken alone, grouped into four general classes: beneficial (green), neutral (blue), deleterious (red), lethal (black).

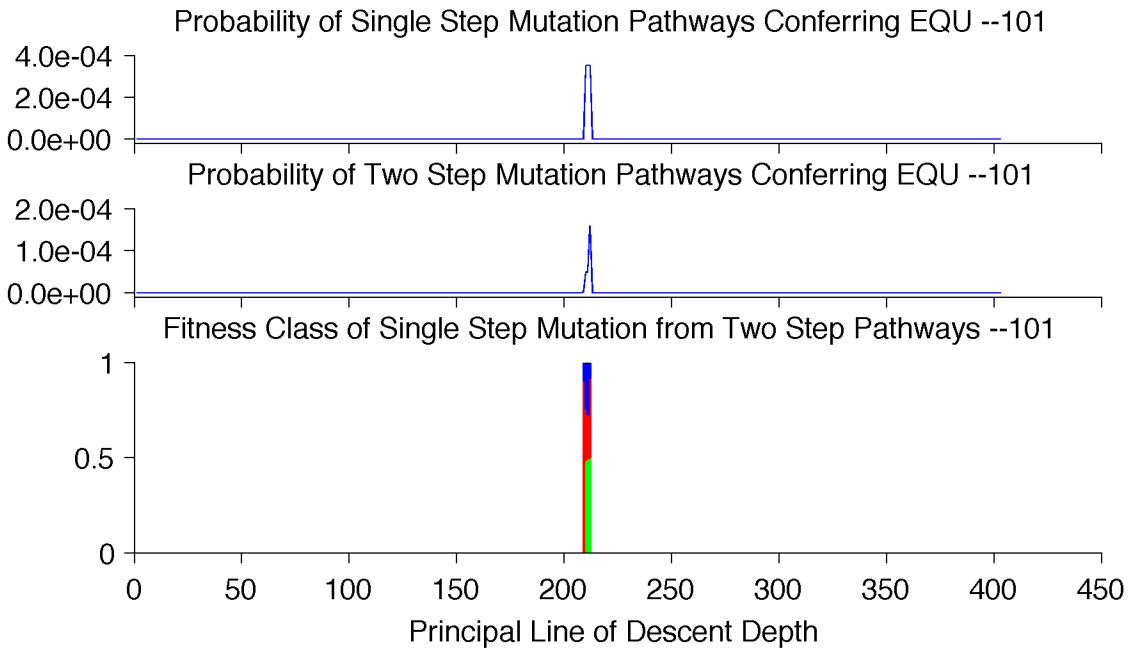

Figure A.1: Mutational neighborhood of the PLoD from population 101
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

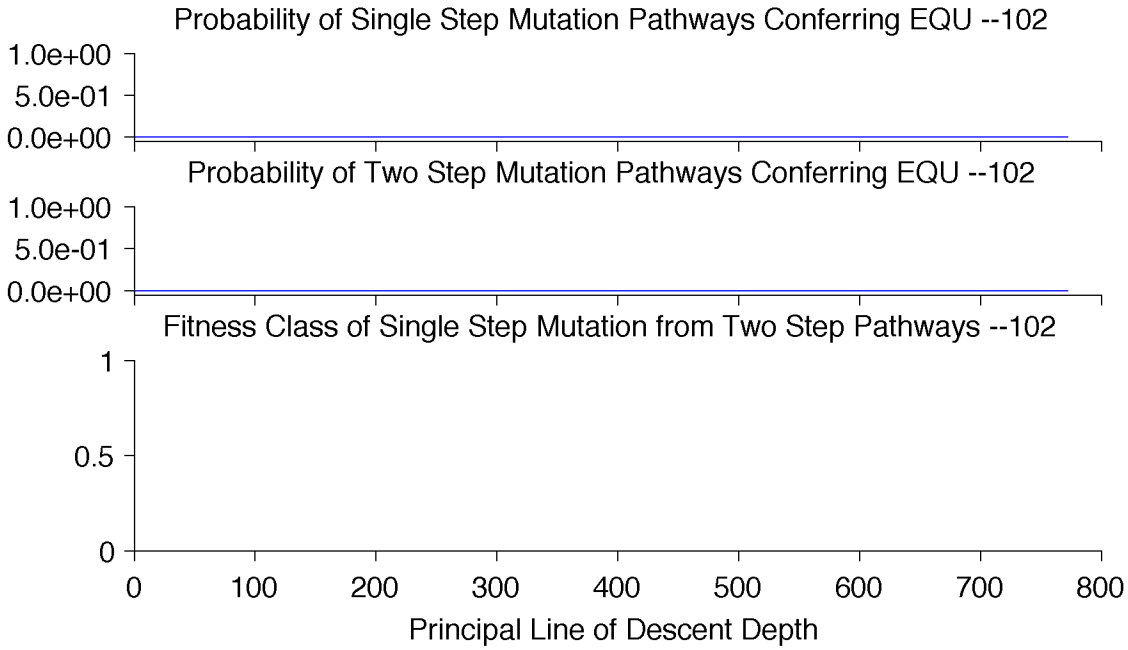

Figure A.2: Mutational neighborhood of the PLoD from population 102
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

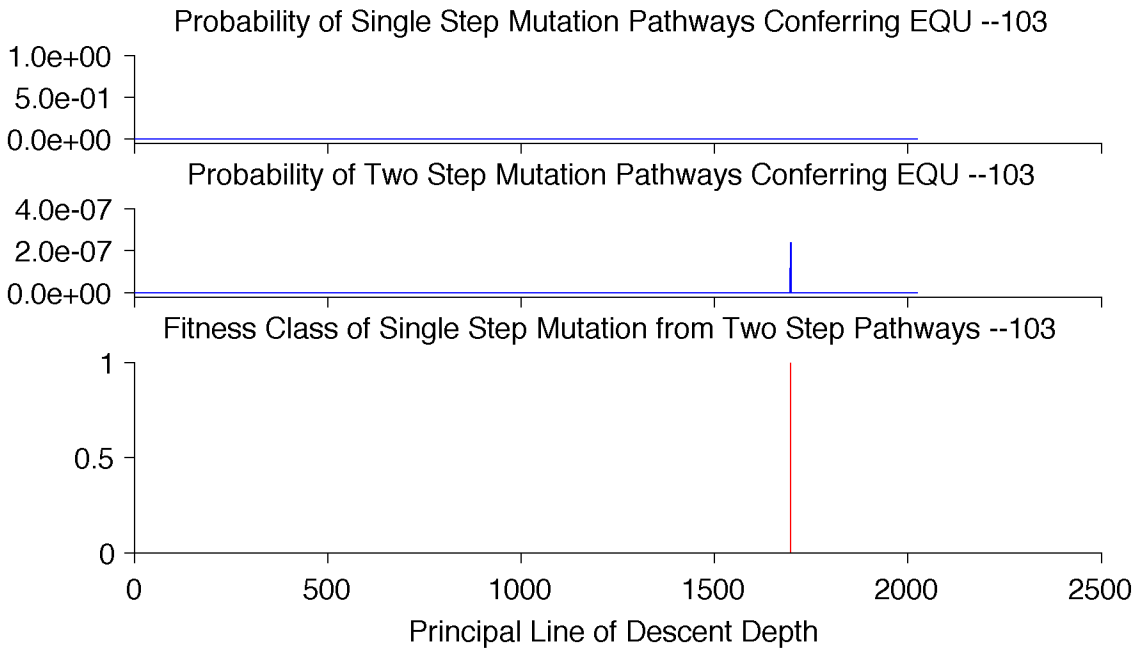

Figure A.3: Mutational neighborhood of the PLoD from population 103
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

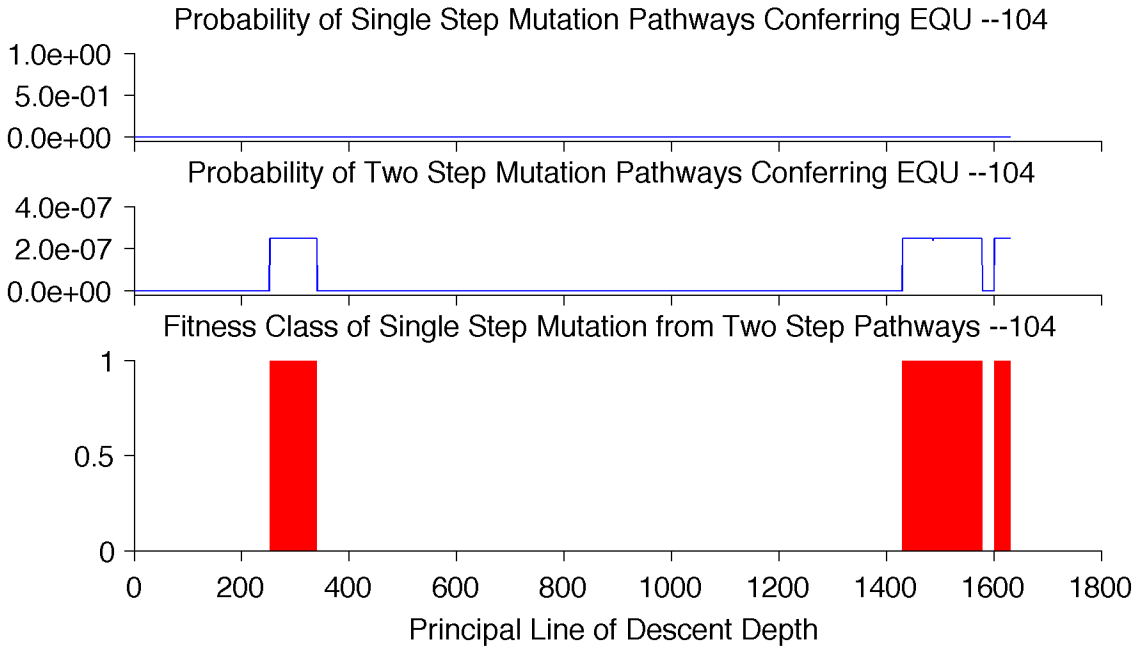

Figure A.4: Mutational neighborhood of the PLoD from population 104
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

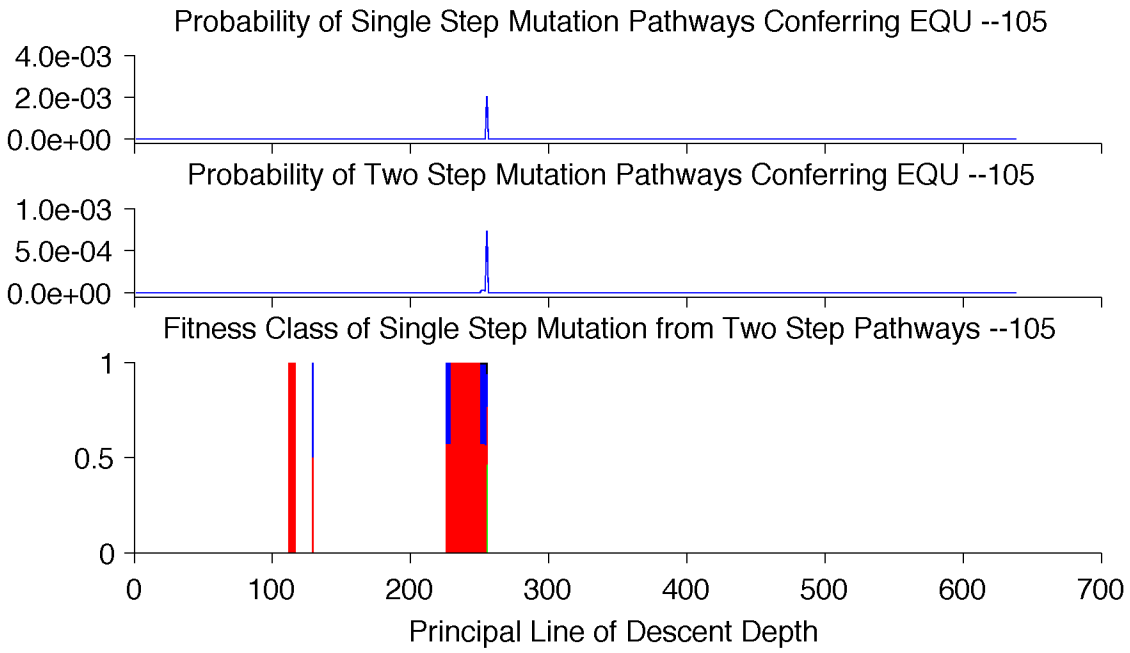

Figure A.5: Mutational neighborhood of the PLoD from population 105
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

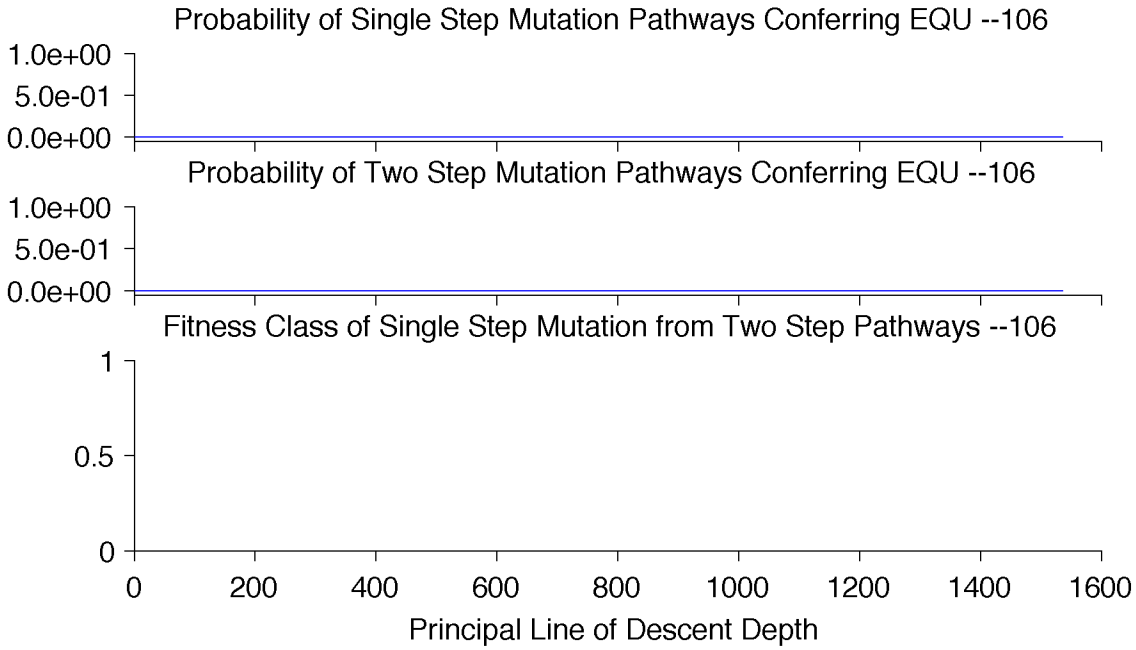

Figure A.6: Mutational neighborhood of the PLoD from population 106
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

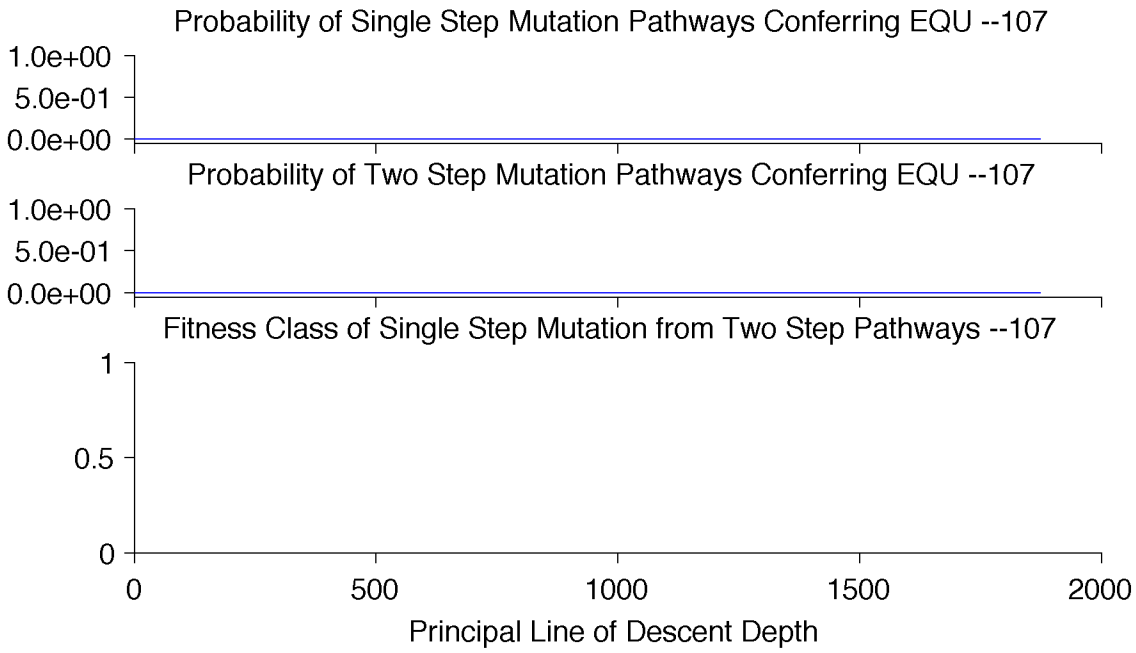

Figure A.7: Mutational neighborhood of the PLoD from population 107
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

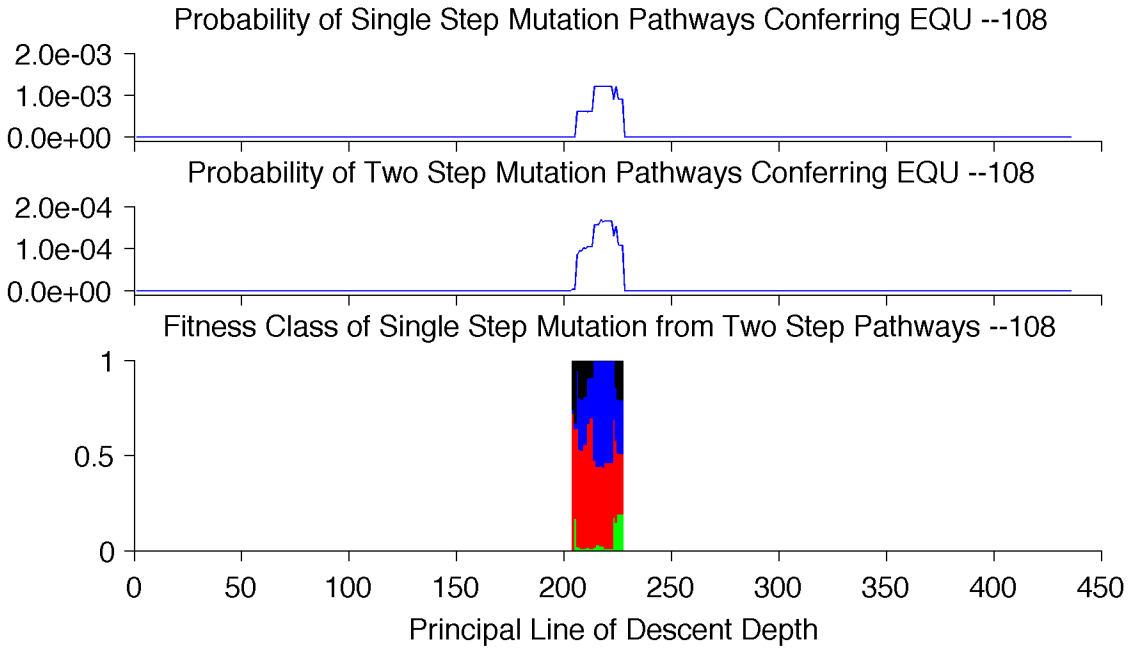

Figure A.8: Mutational neighborhood of the PLoD from population 108
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

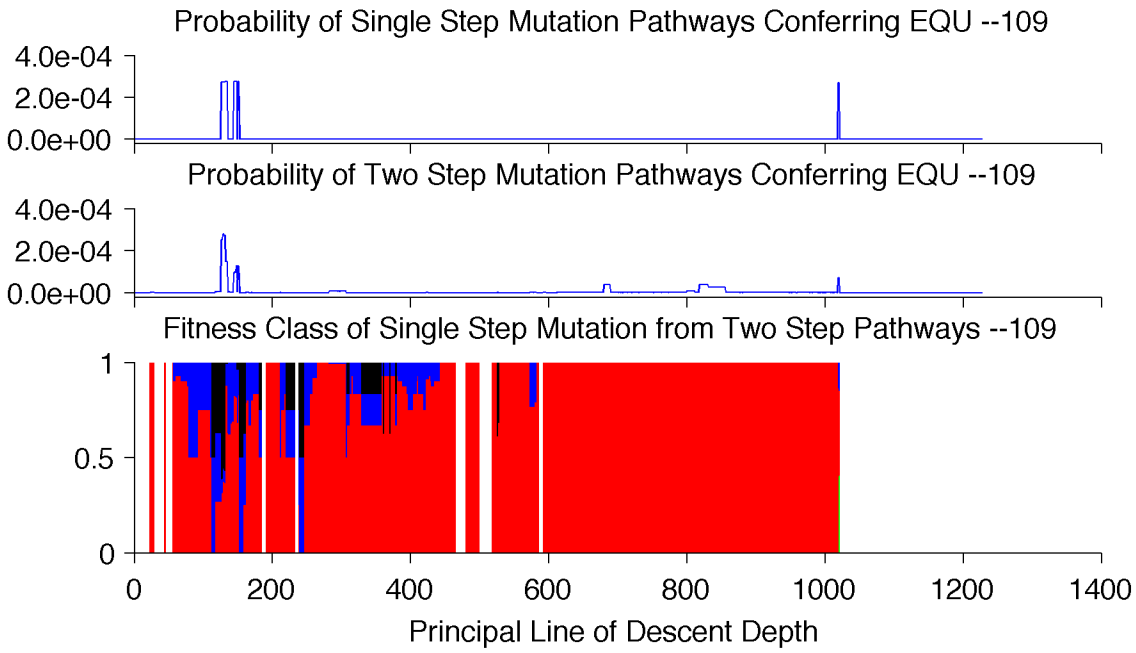

Figure A.9: Mutational neighborhood of the PLoD from population 109
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

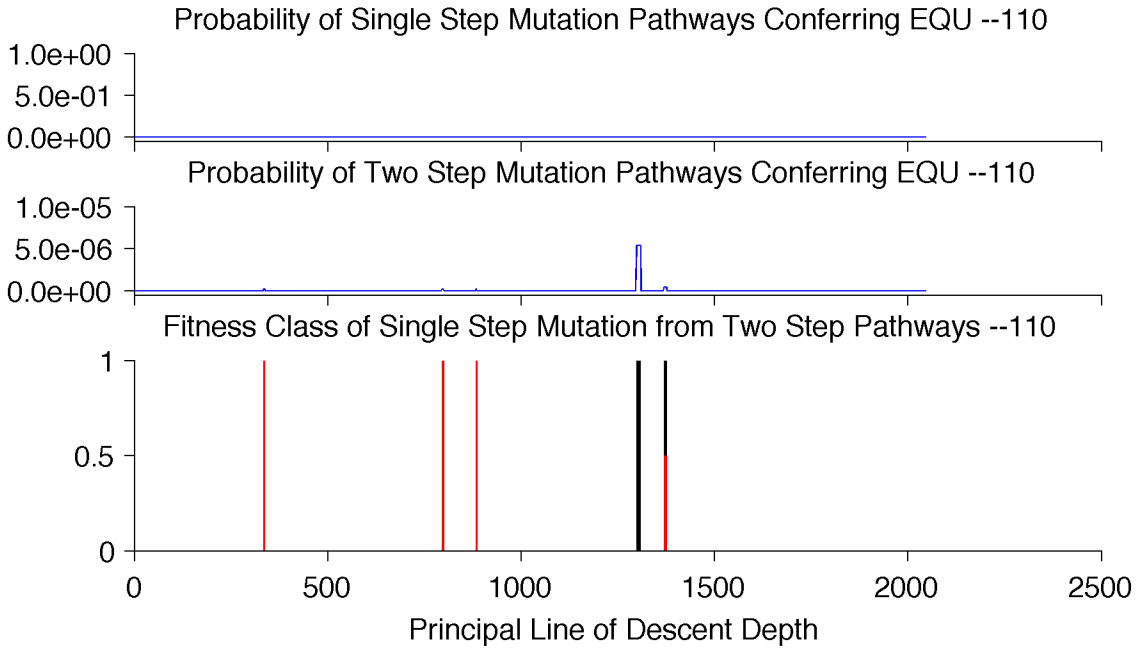

Figure A.10: Mutational neighborhood of the PLoD from population 110
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

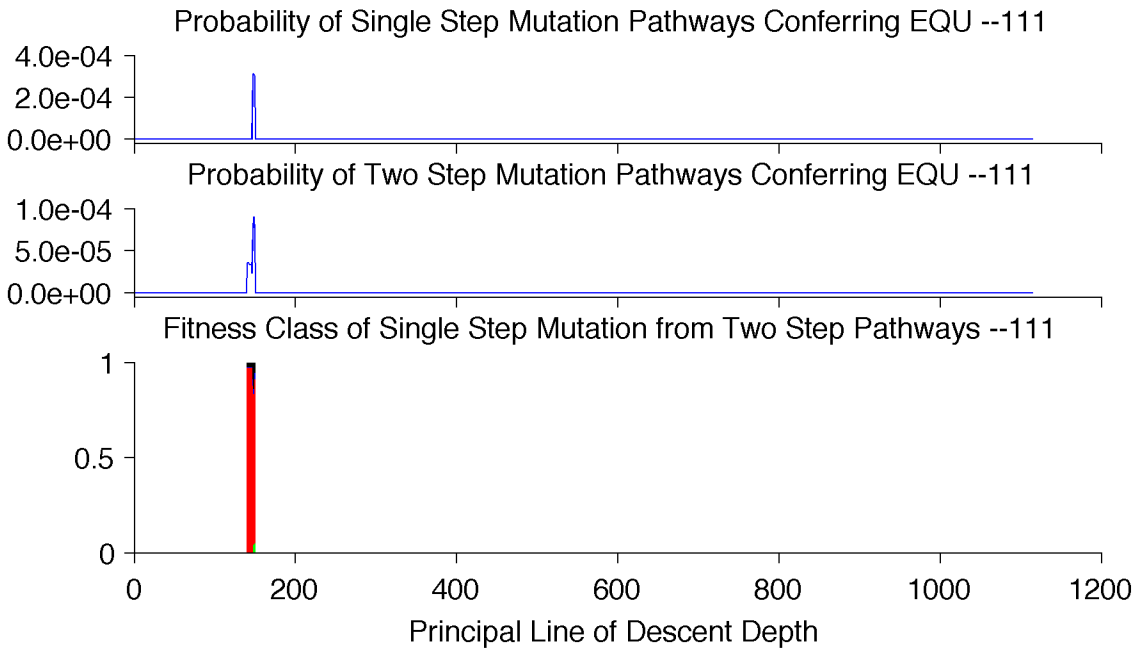

Figure A.11: Mutational neighborhood of the PLoD from population 111
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

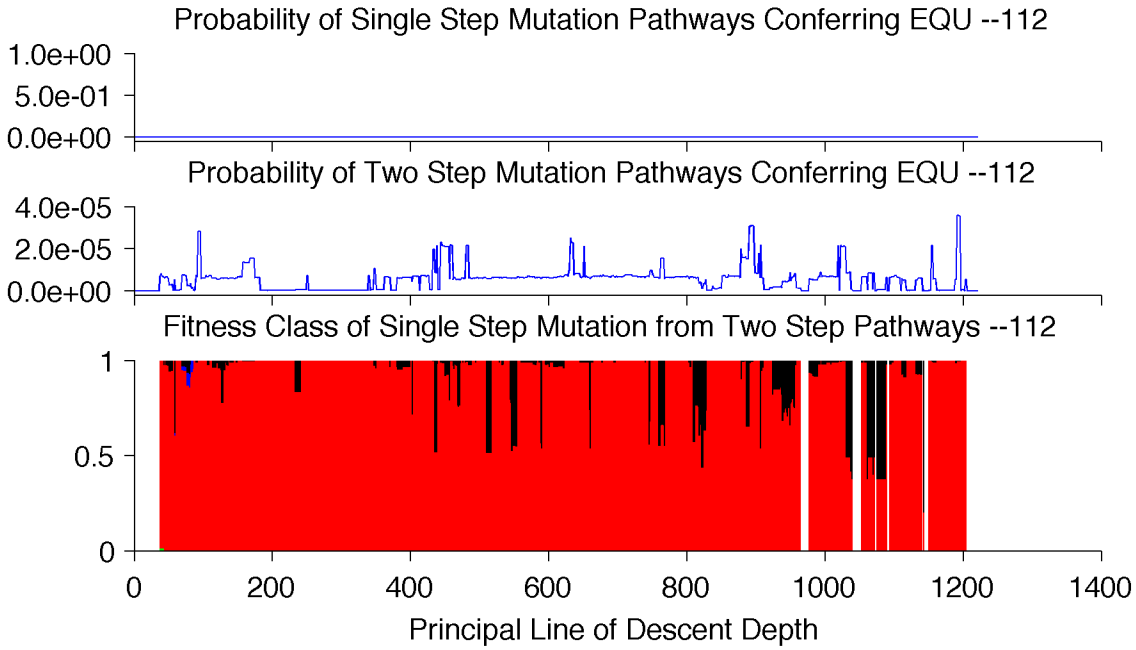

Figure A.12: Mutational neighborhood of the PLoD from population 112
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

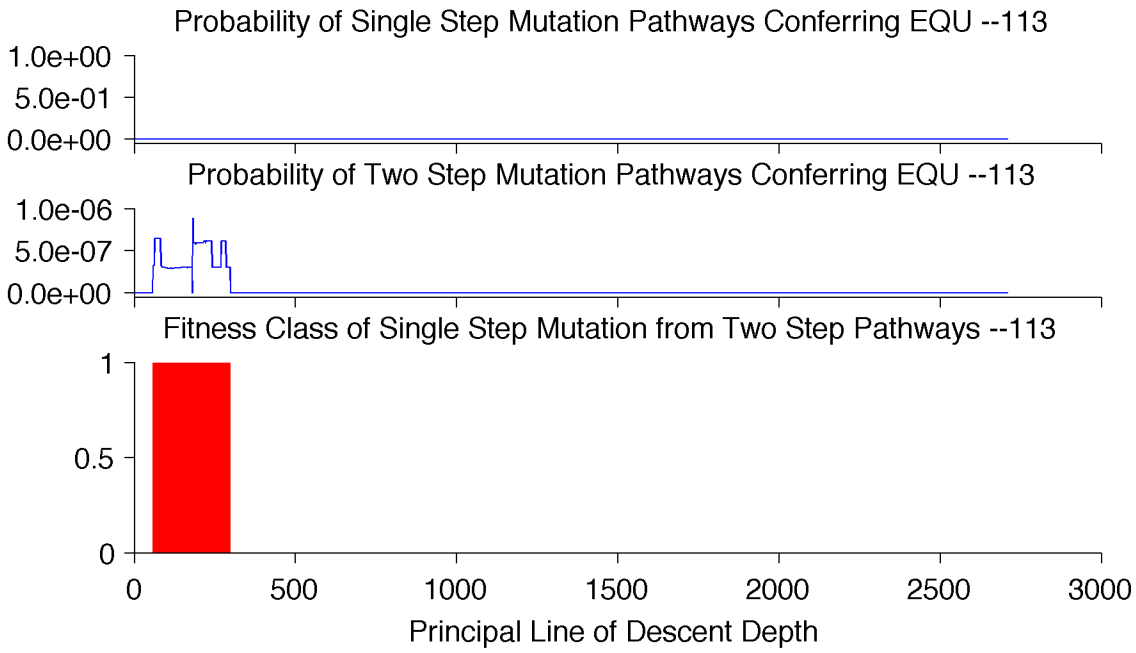

Figure A.13: Mutational neighborhood of the PLoD from population 113
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

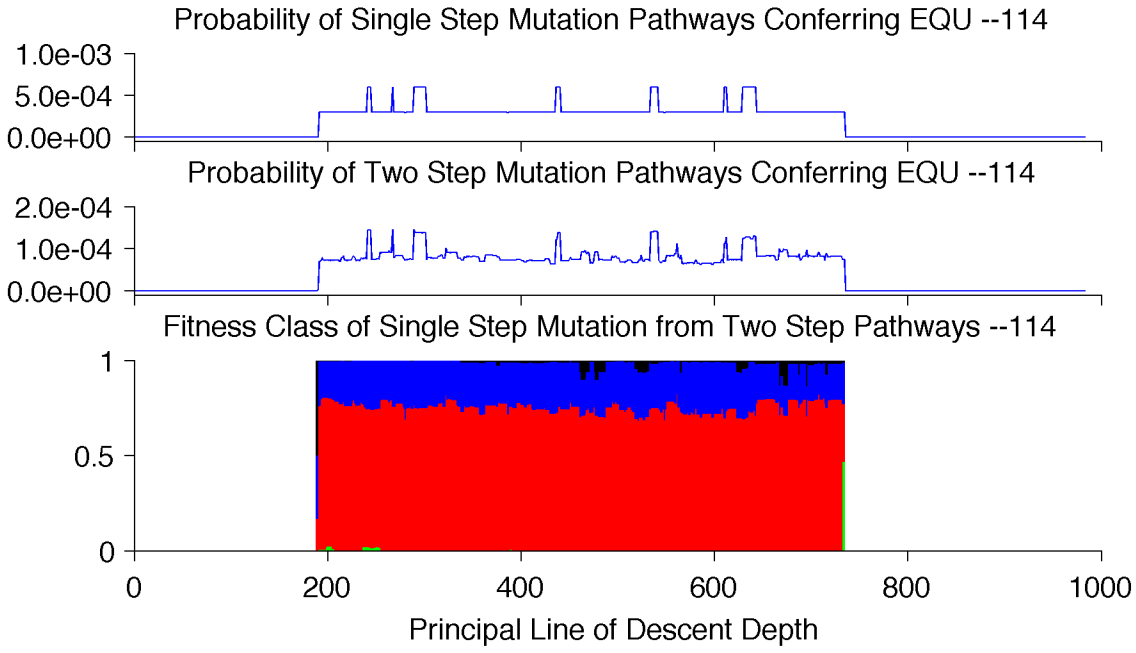

Figure A.14: Mutational neighborhood of the PLoD from population 114
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

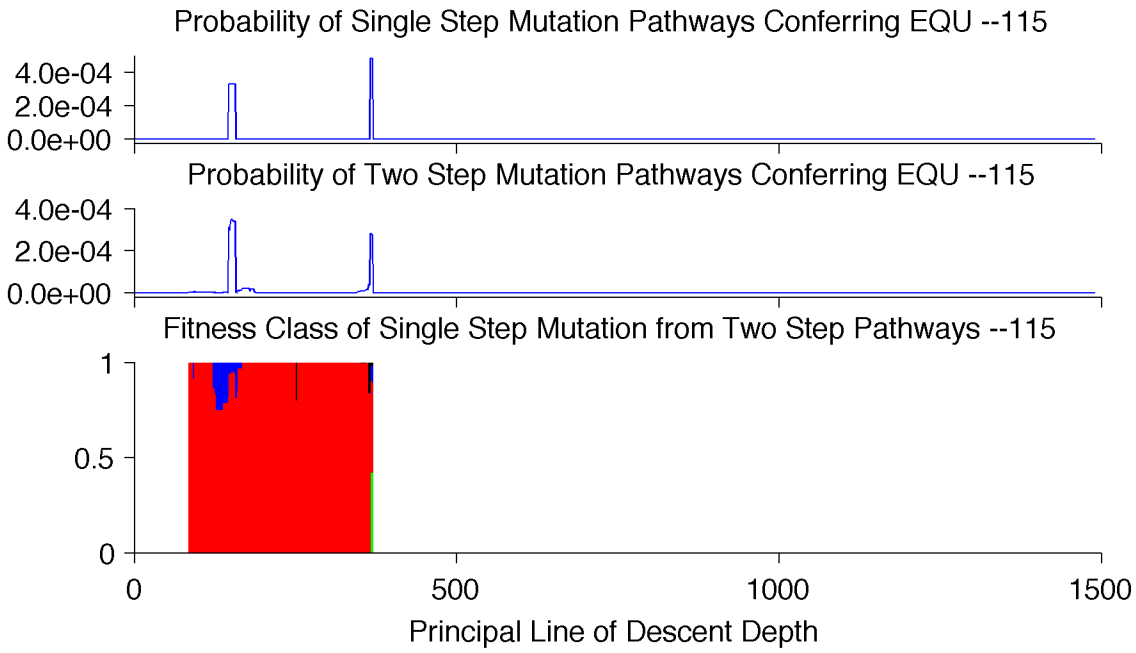

Figure A.15: Mutational neighborhood of the PLoD from population 115
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

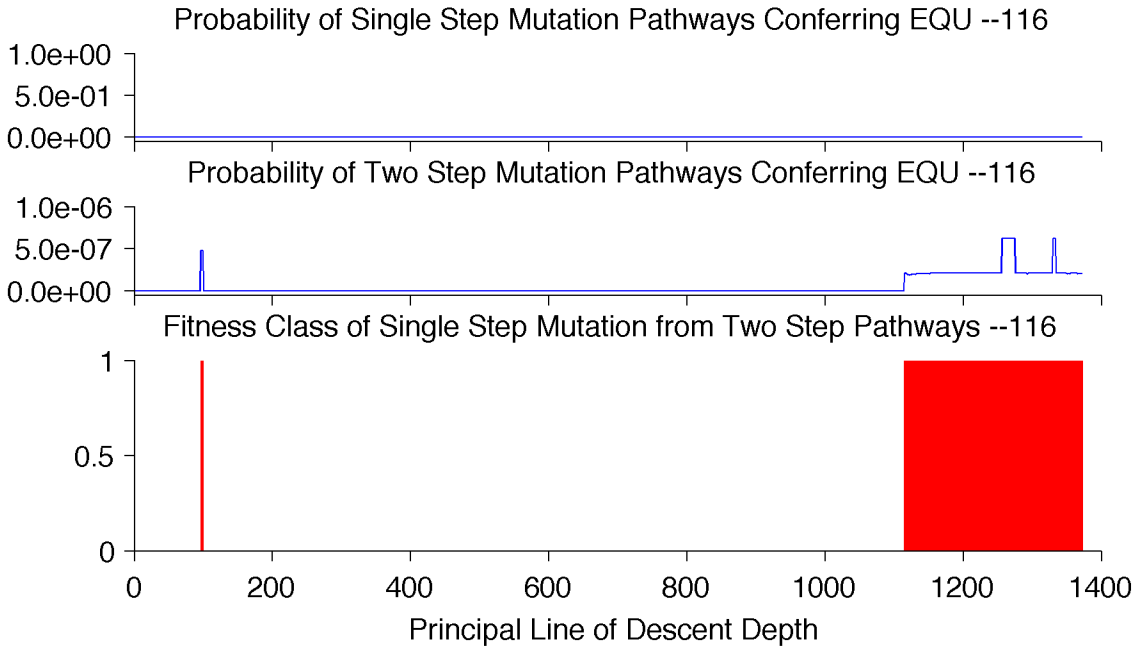

Figure A.16: Mutational neighborhood of the PLoD from population 116
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

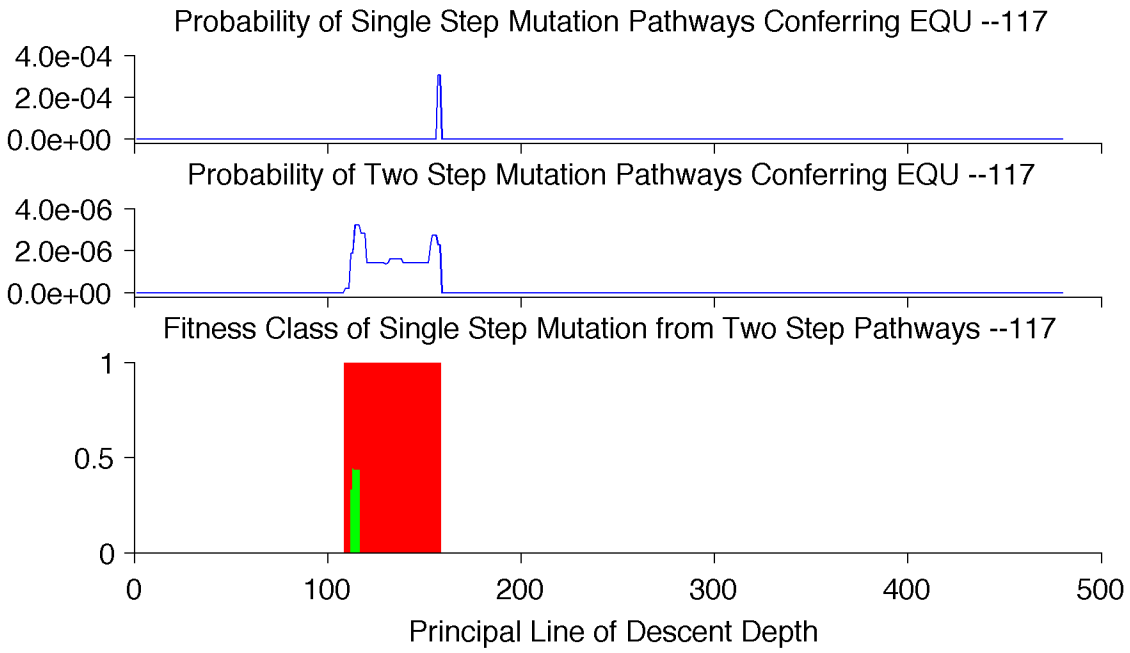

Figure A.17: Mutational neighborhood of the PLoD from population 117
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

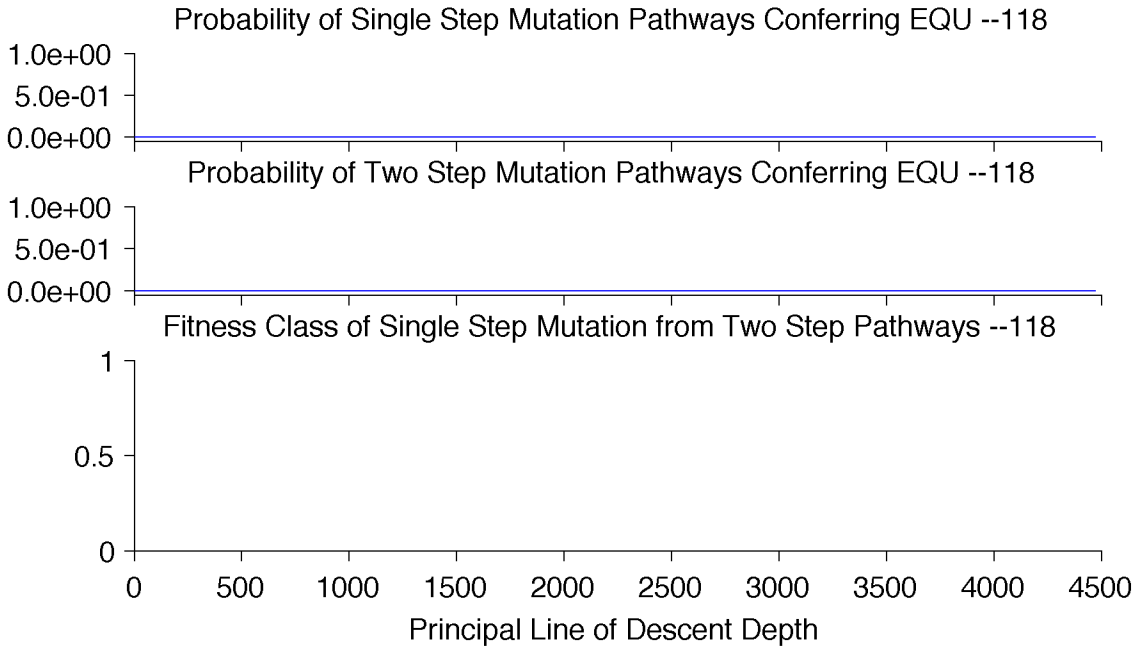

Figure A.18: Mutational neighborhood of the PLoD from population 118
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

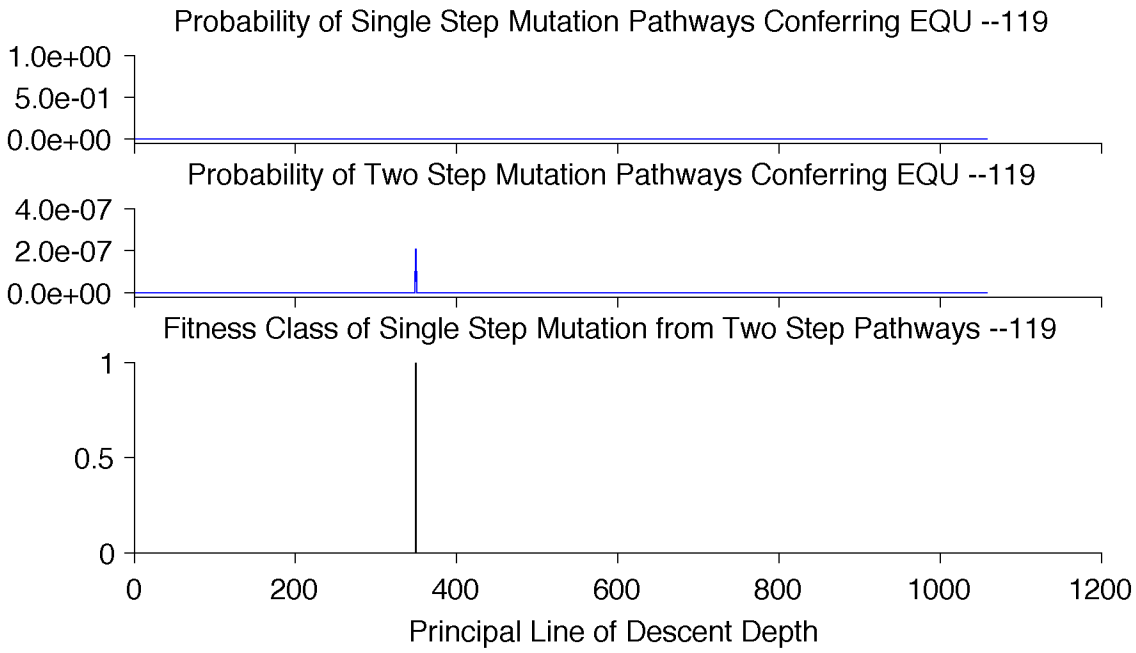

Figure A.19: Mutational neighborhood of the PLoD from population 119
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

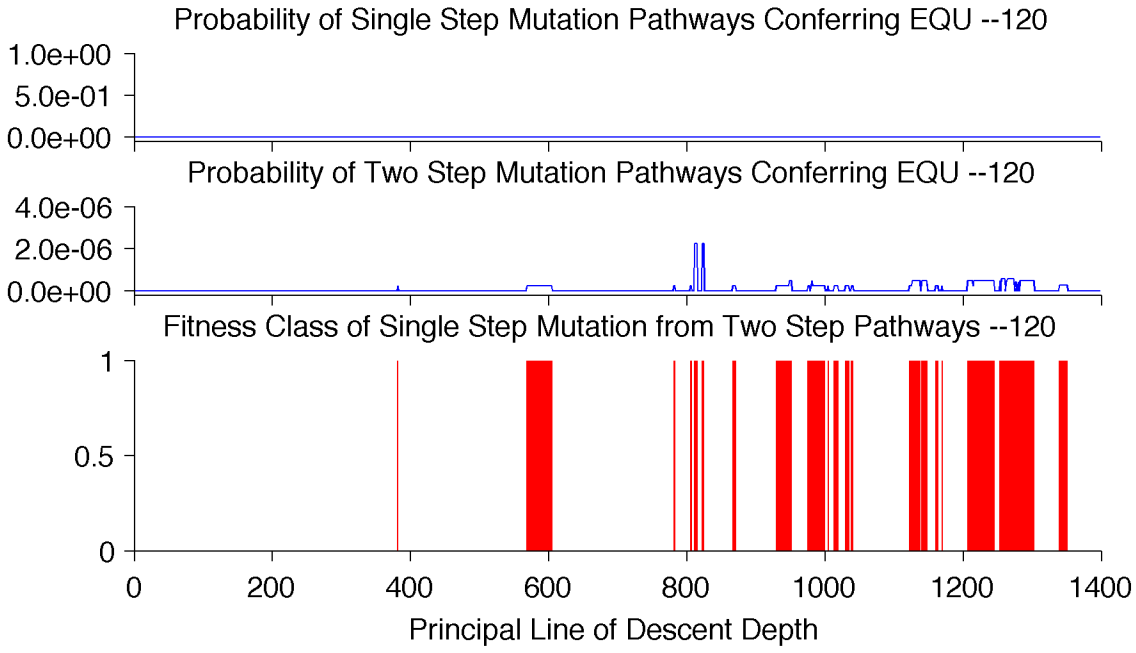

Figure A.20: Mutational neighborhood of the PLoD from population 120
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

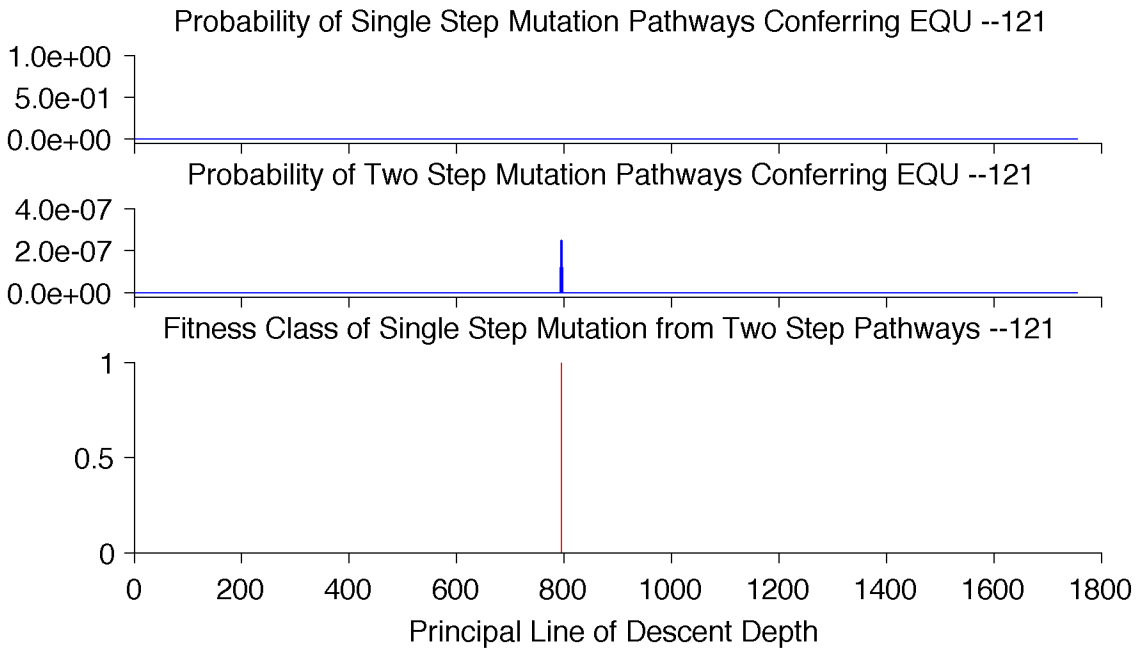

Figure A.21: Mutational neighborhood of the PLoD from population 121
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

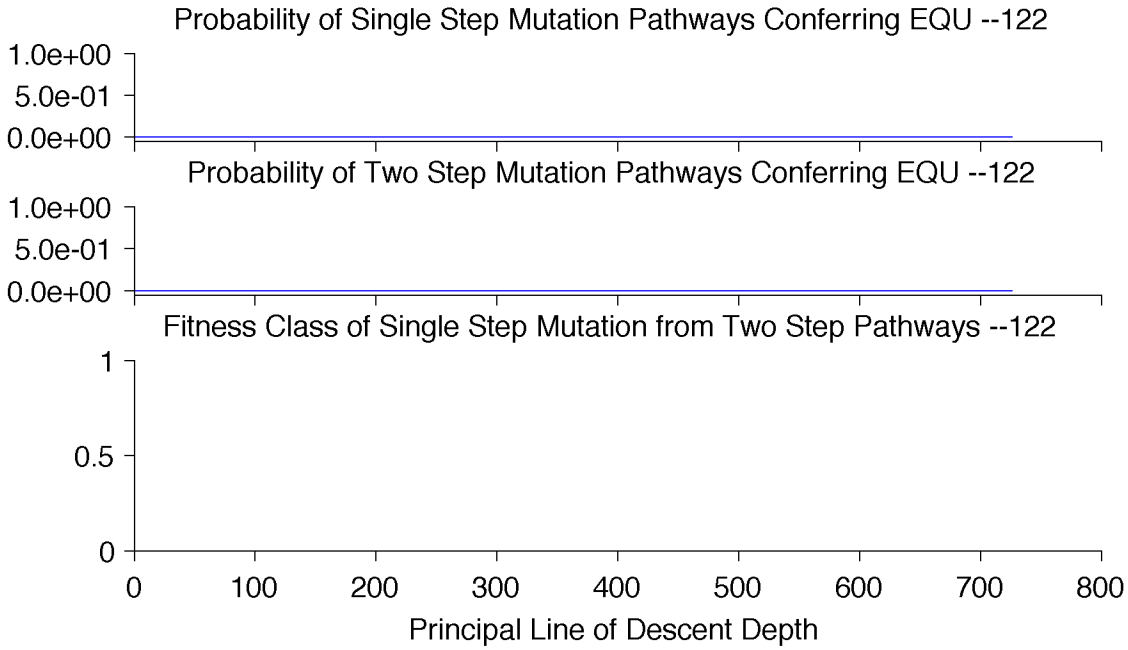

Figure A.22: Mutational neighborhood of the PLoD from population 122
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

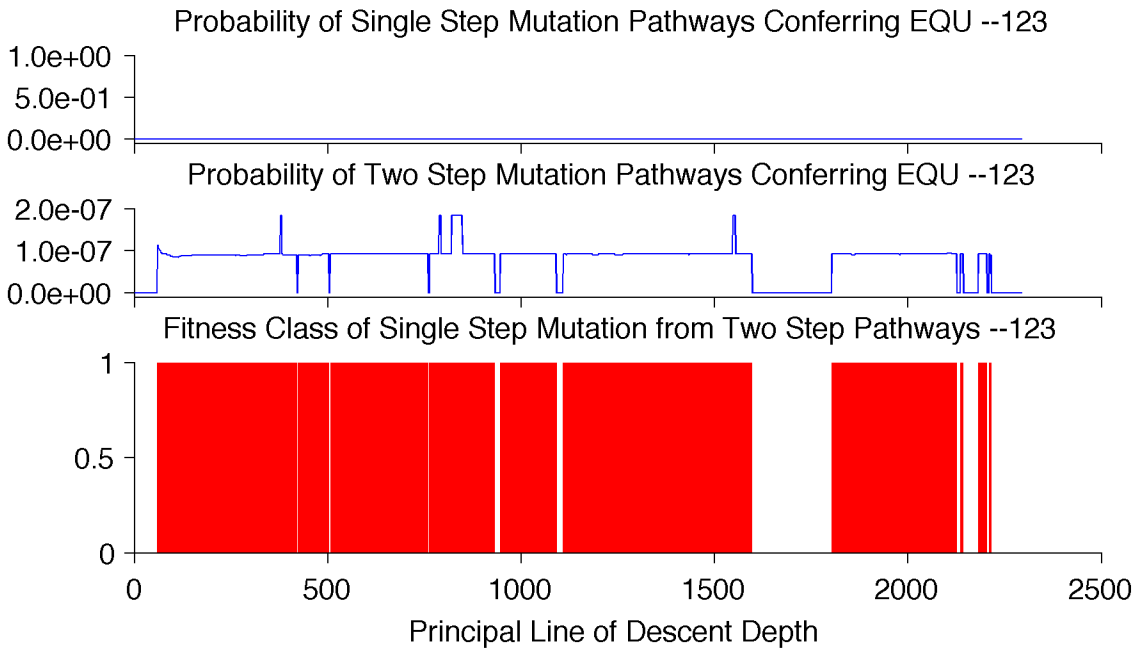

Figure A.23: Mutational neighborhood of the PLoD from population 123
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

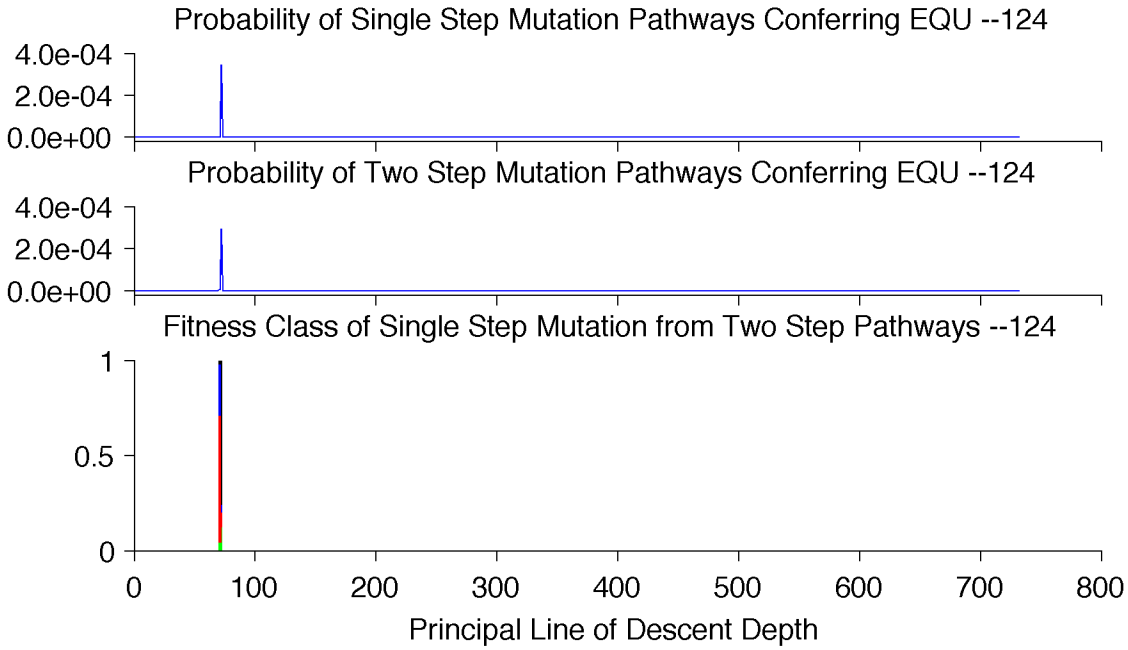

Figure A.24: Mutational neighborhood of the PLoD from population 124
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

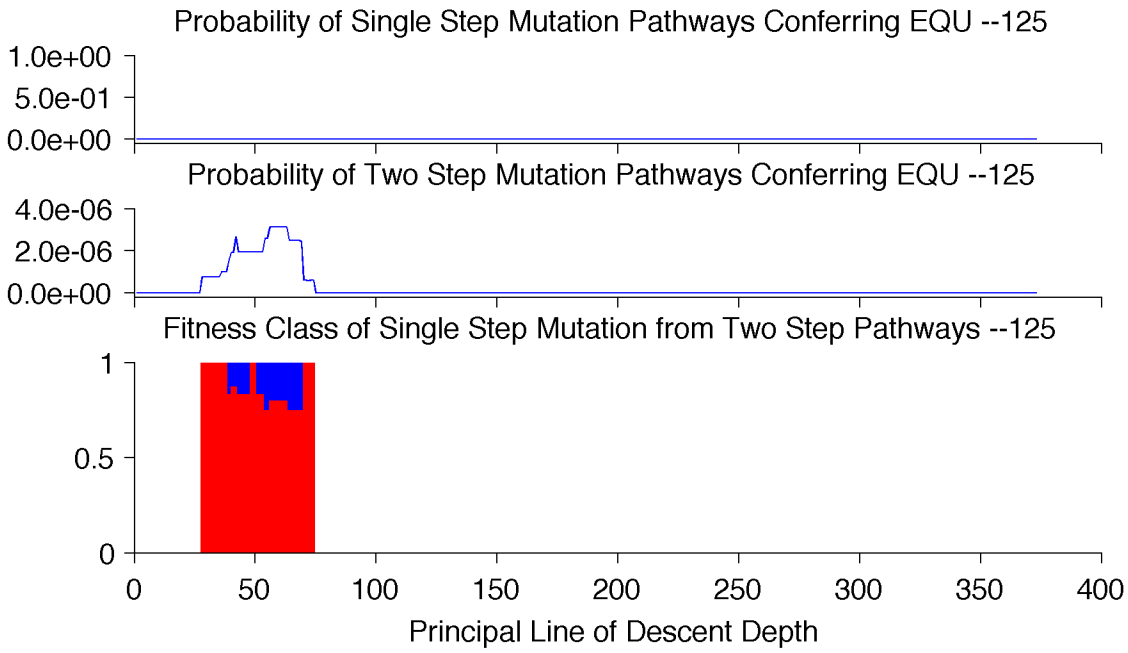

Figure A.25: Mutational neighborhood of the PLoD from population 125
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

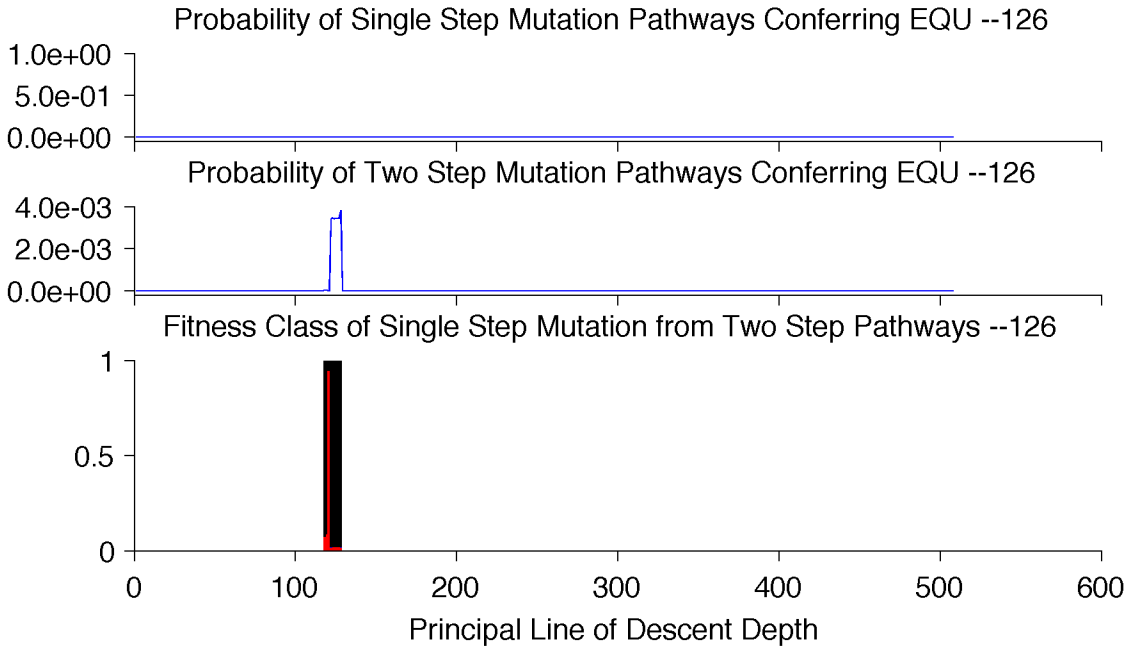

Figure A.26: Mutational neighborhood of the PLoD from population 126
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

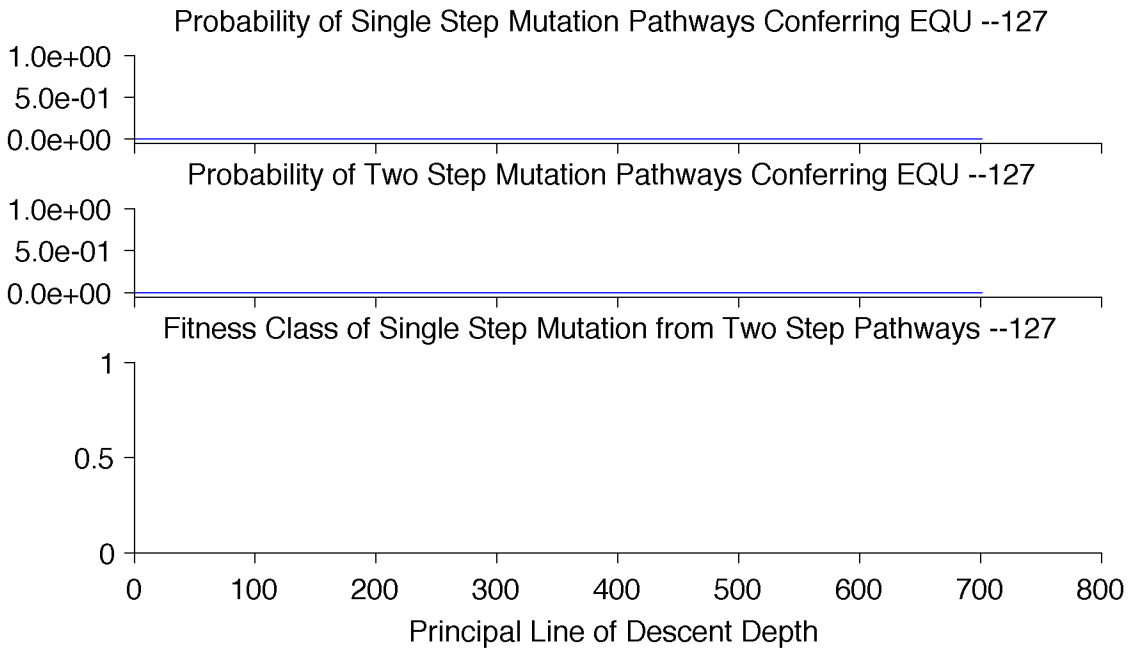

Figure A.27: Mutational neighborhood of the PLoD from population 127
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

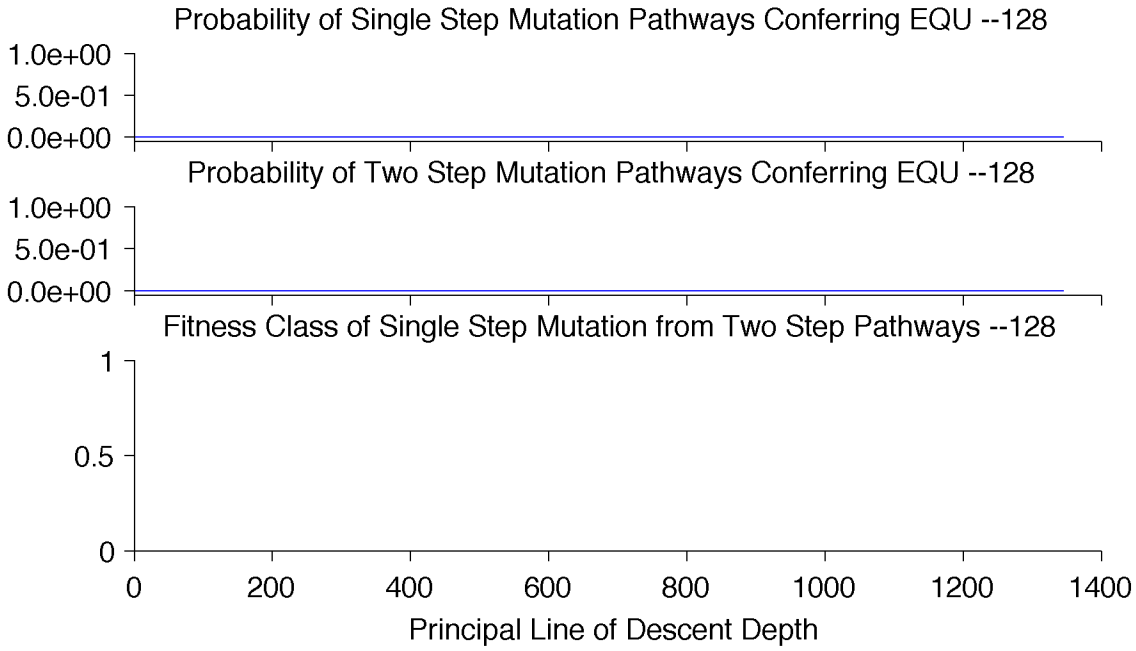

Figure A.28: Mutational neighborhood of the PLoD from population 128
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

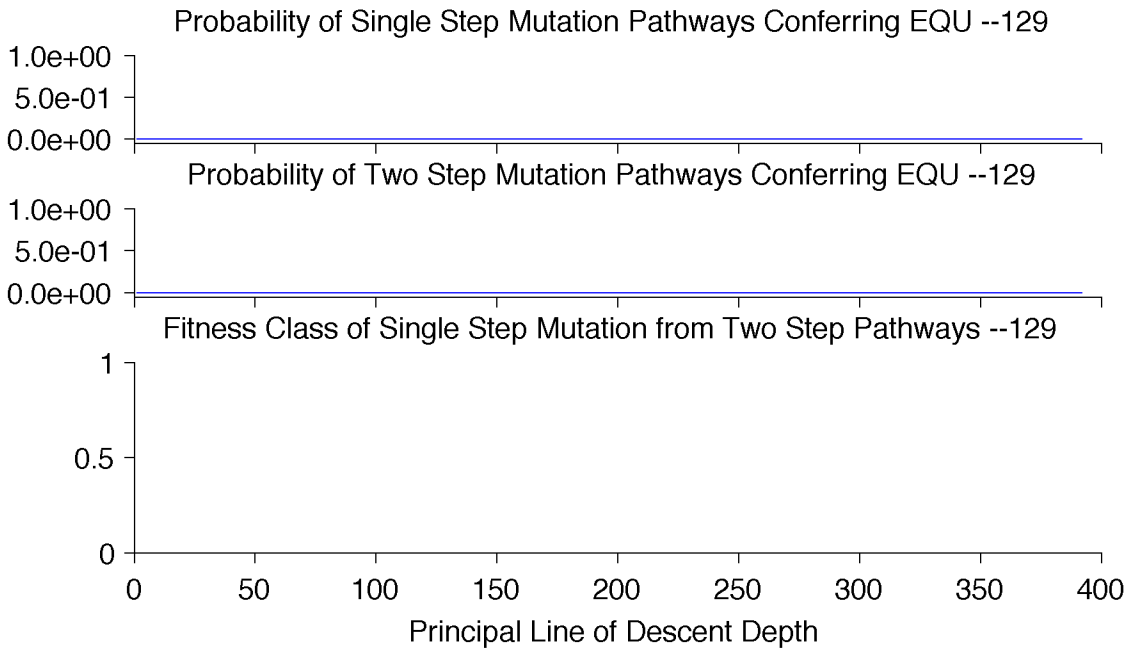

Figure A.29: Mutational neighborhood of the PLoD from population 129
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

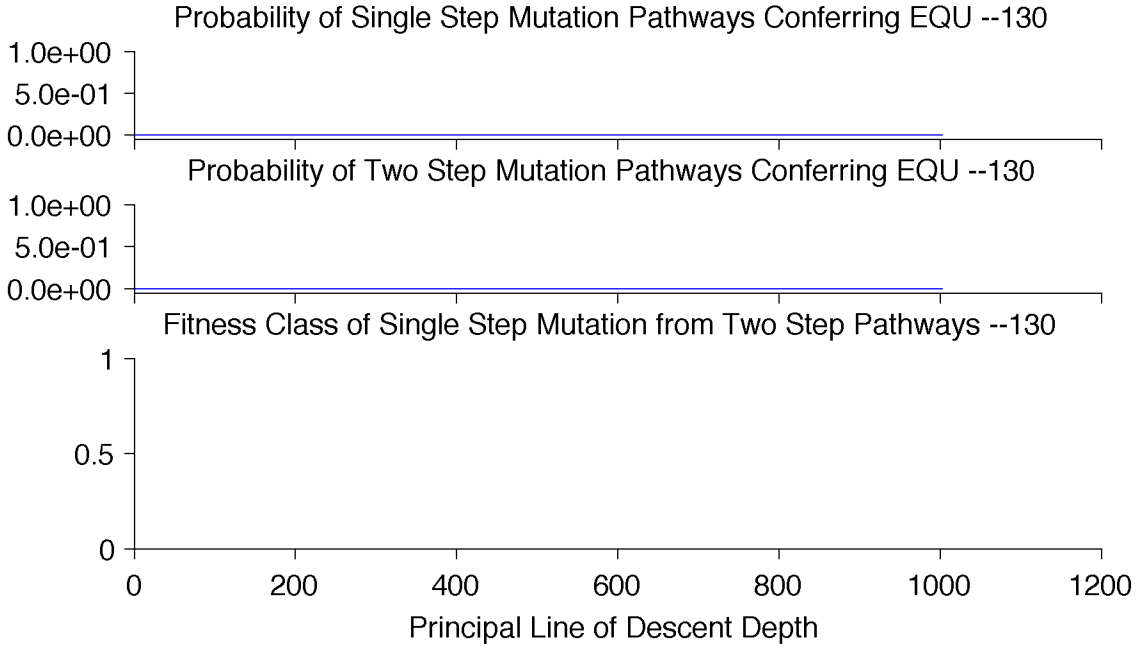

Figure A.30: Mutational neighborhood of the PLoD from population 130
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

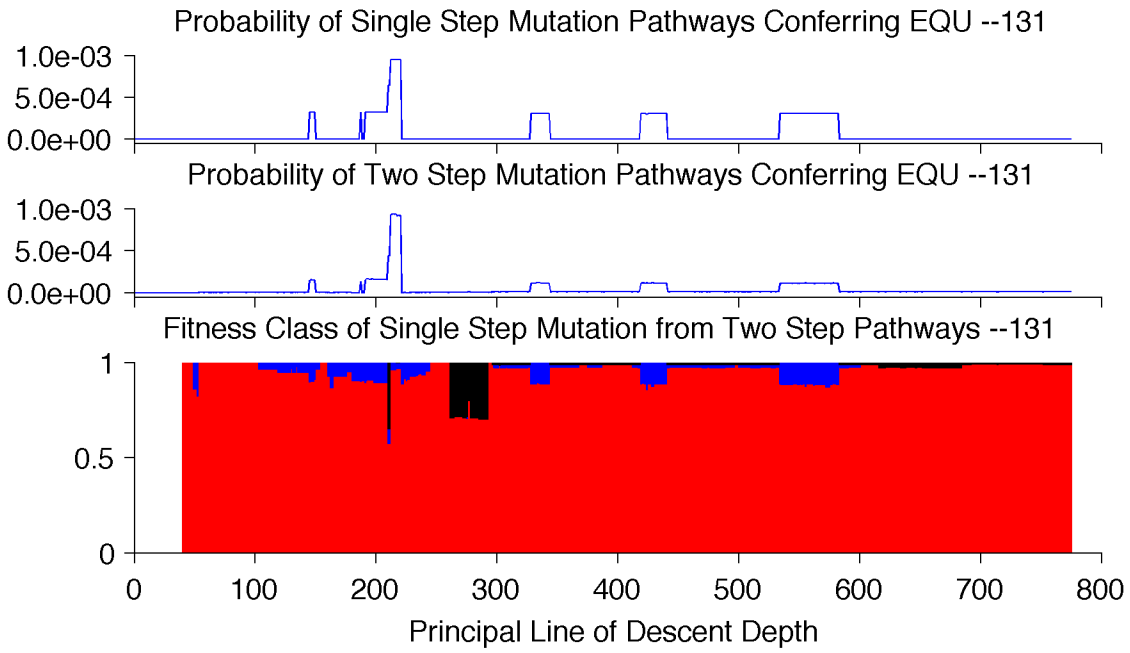

Figure A.31: Mutational neighborhood of the PLoD from population 132
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

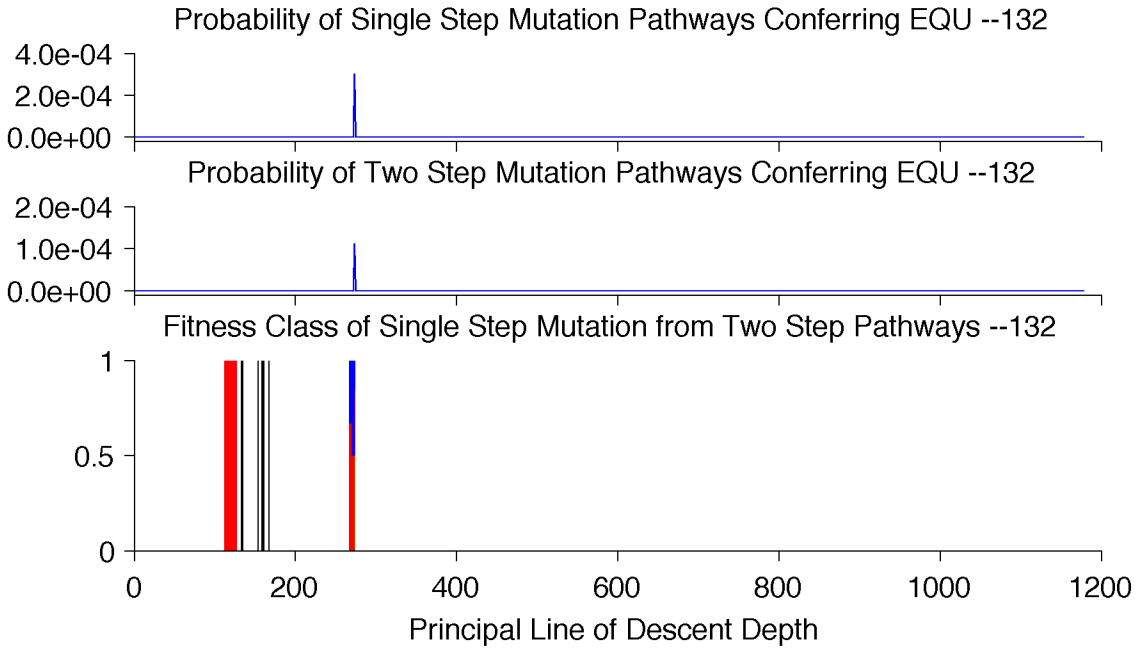

Figure A.32: Mutational neighborhood of the PLoD from population 132
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

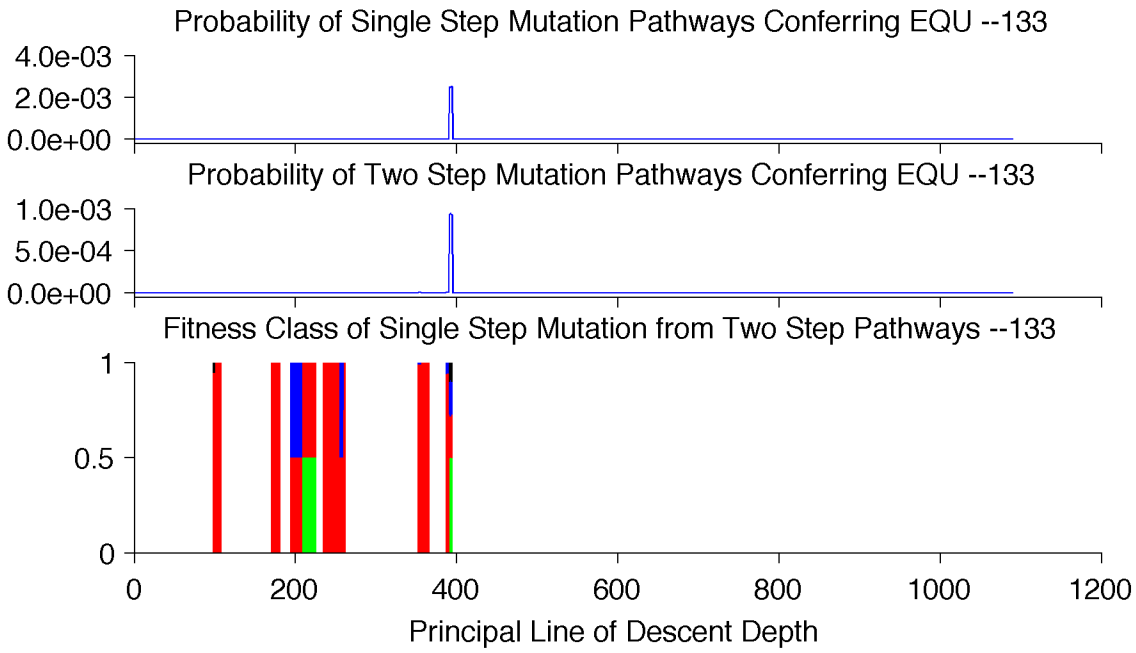

Figure A.33: Mutational neighborhood of the PLoD from population 133
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

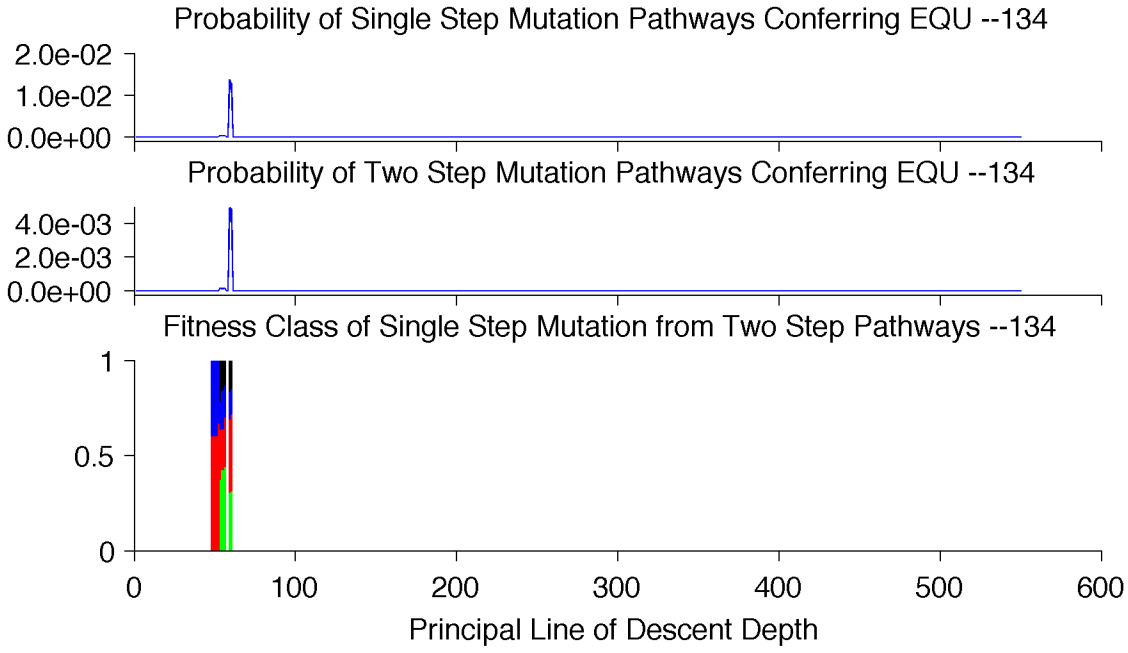

Figure A.34: Mutational neighborhood of the PLoD from population 134
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

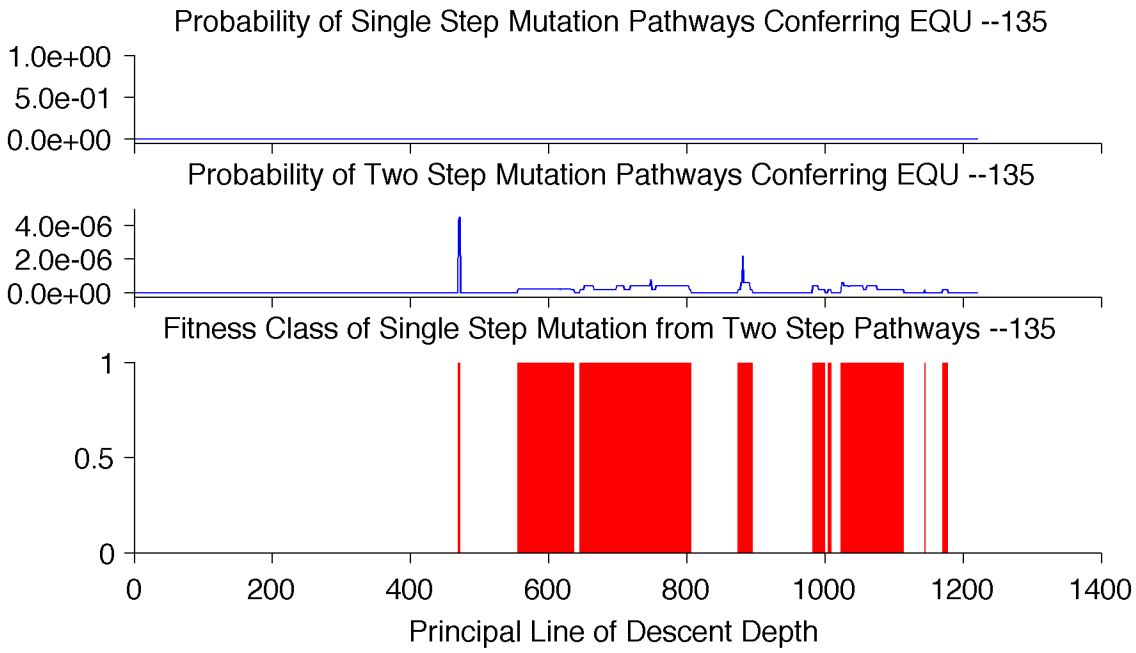

Figure A.35: Mutational neighborhood of the PLoD from population 135
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

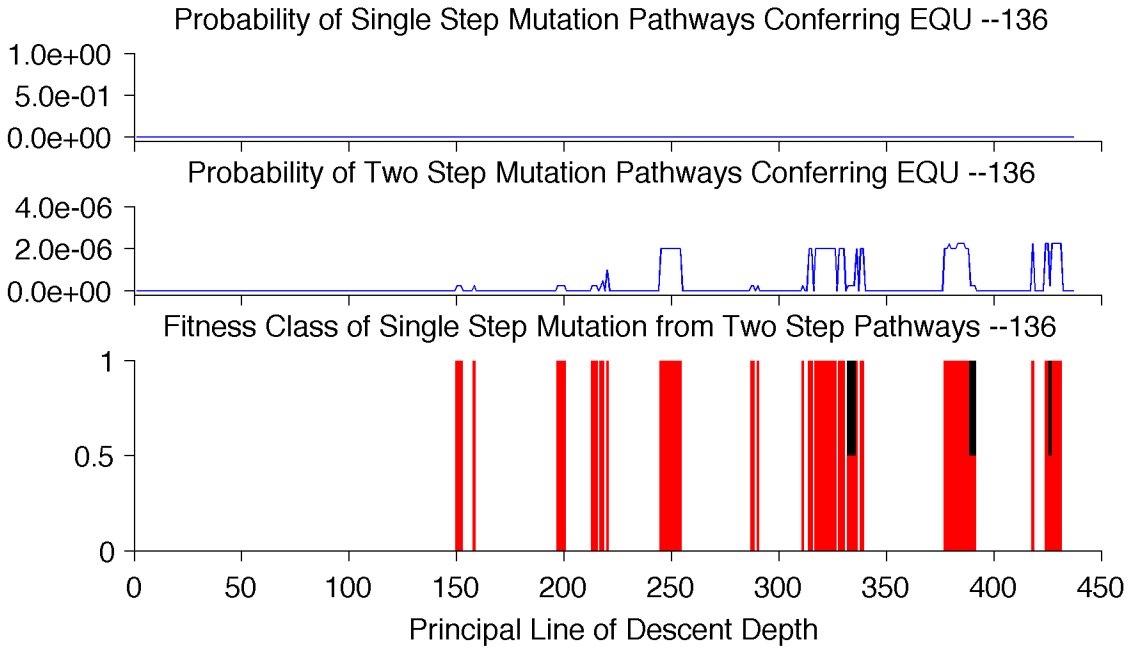

Figure A.36: Mutational neighborhood of the PLoD from population 136
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

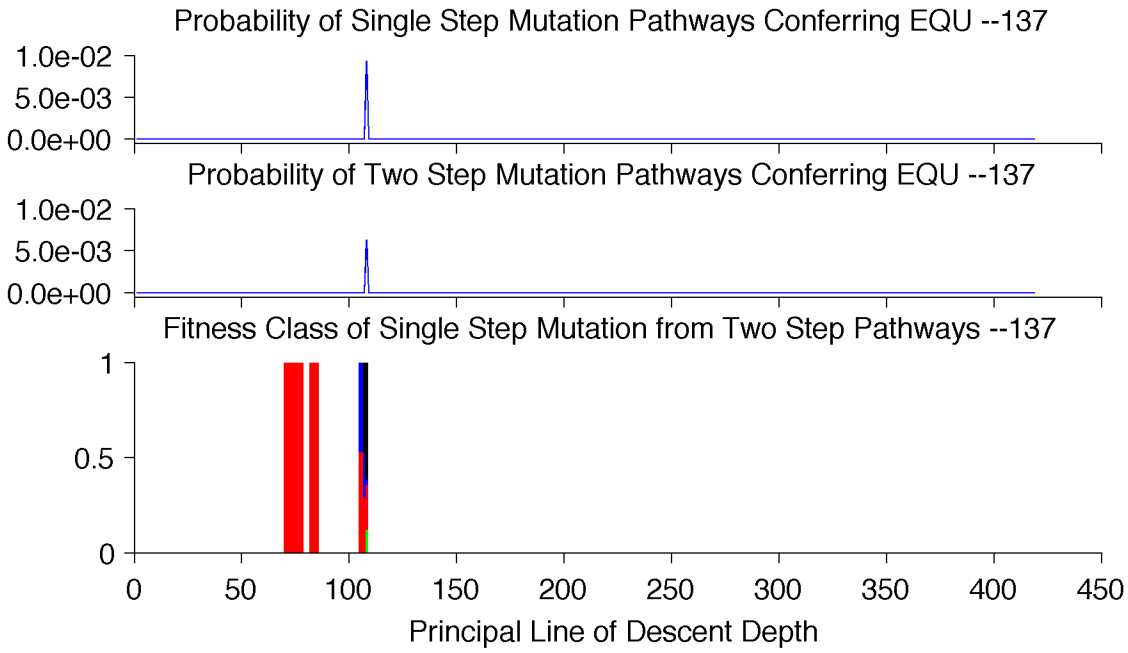

Figure A.37: Mutational neighborhood of the PLoD from population 137
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

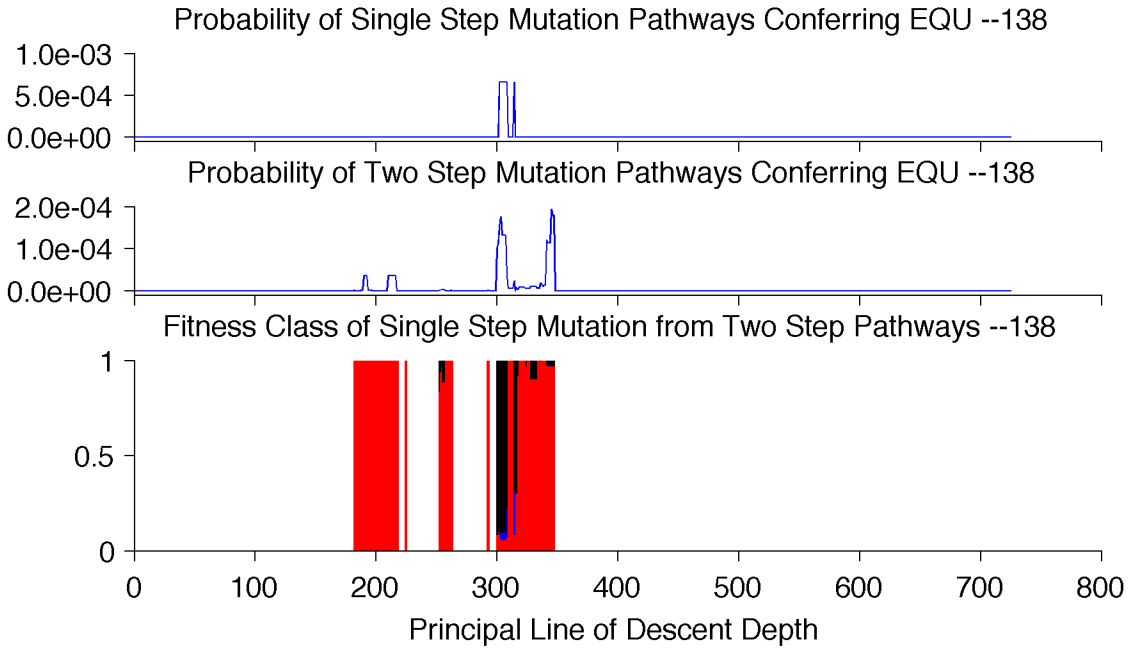

Figure A.38: Mutational neighborhood of the PLoD from population 138
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

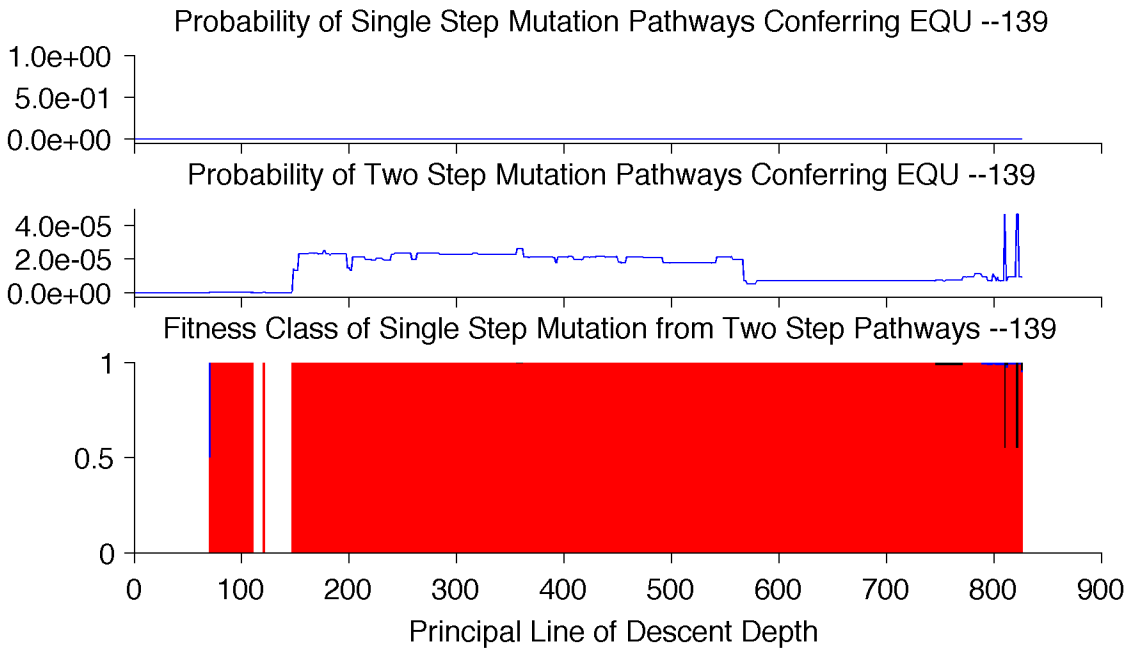

Figure A.39: Mutational neighborhood of the PLoD from population 139
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

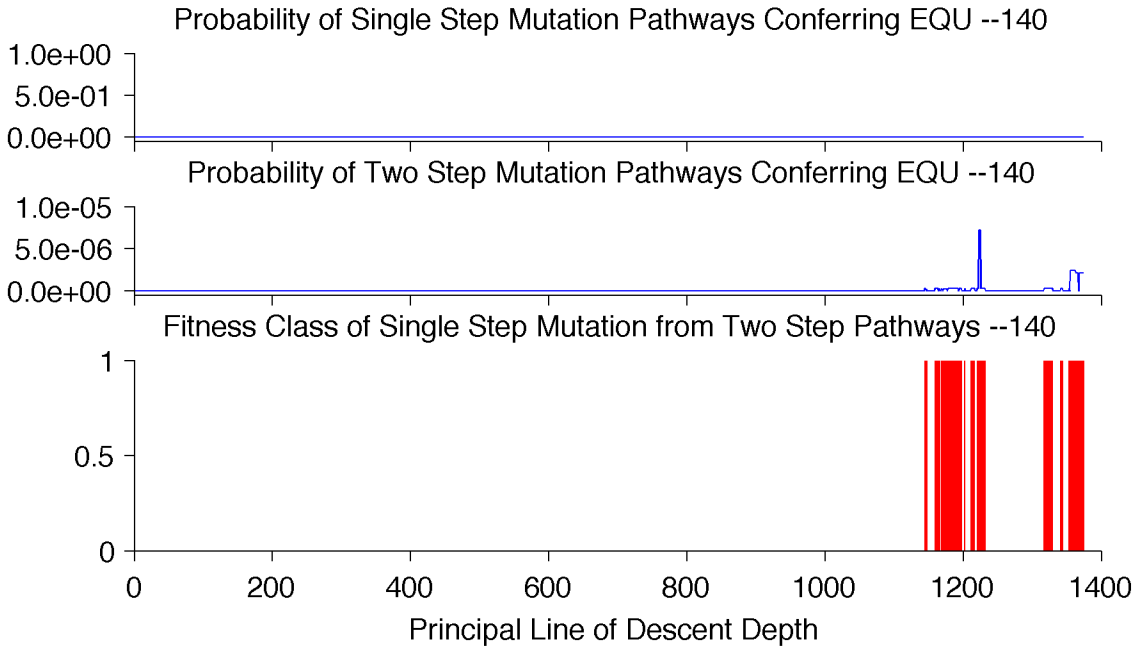

Figure A.40: Mutational neighborhood of the PLoD from population 140
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

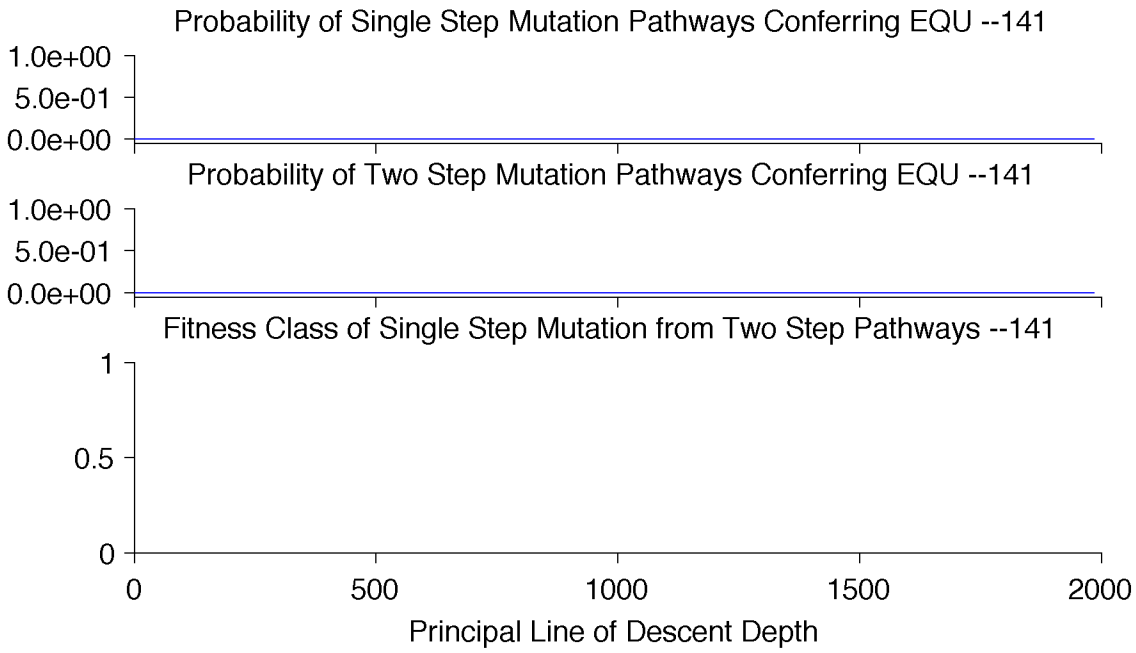

Figure A.41: Mutational neighborhood of the PLoD from population 141
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

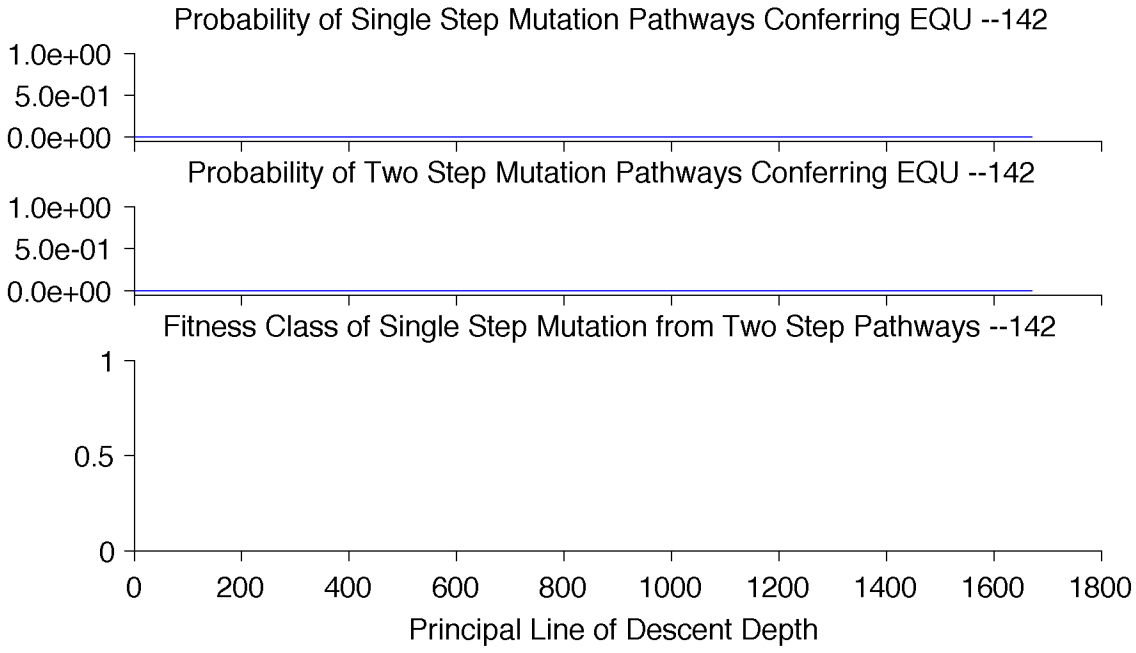

Figure A.42: Mutational neighborhood of the PLoD from population 142
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

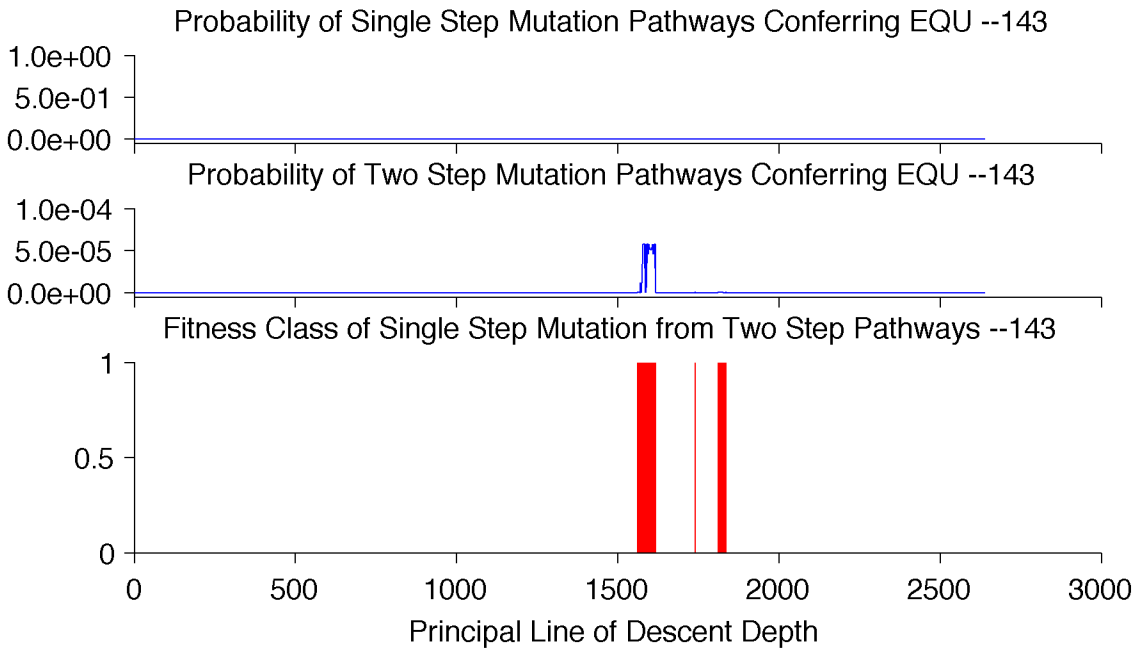

Figure A.43: Mutational neighborhood of the PLoD from population 143
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

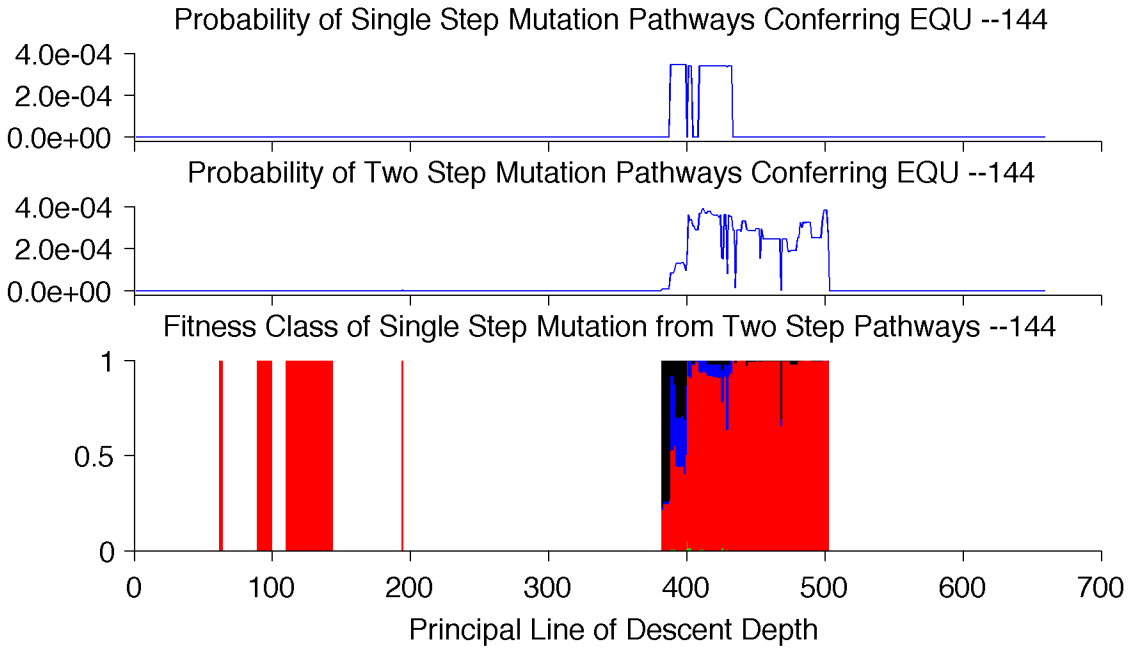

Figure A.44: Mutational neighborhood of the PLoD from population 144
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

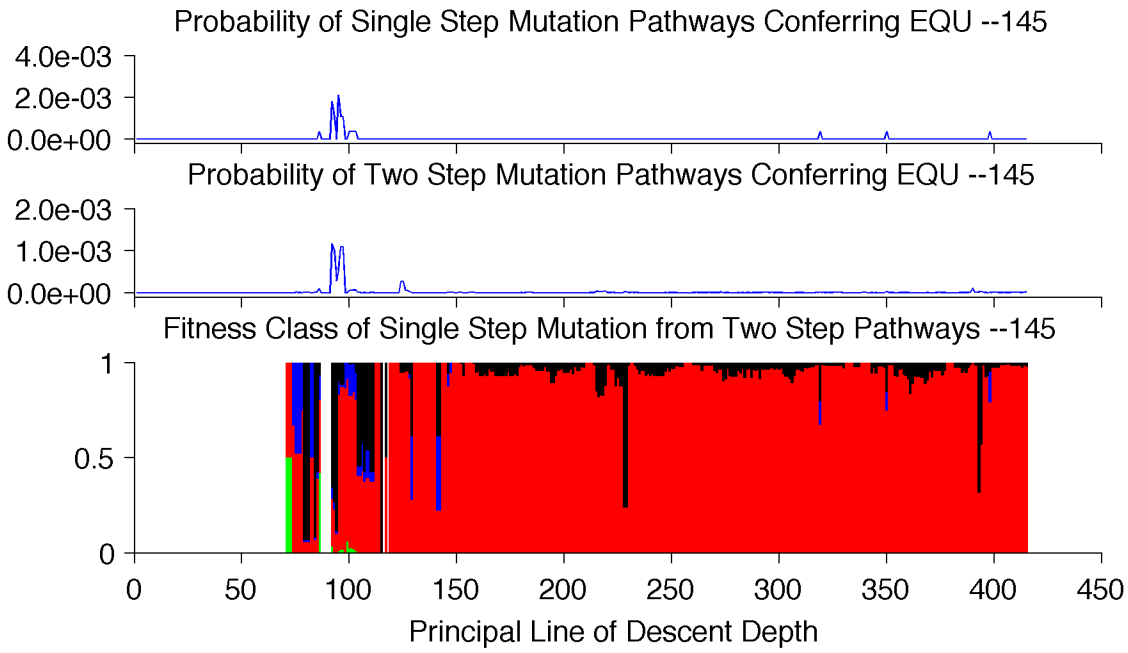

Figure A.45: Mutational neighborhood of the PLoD from population 145
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

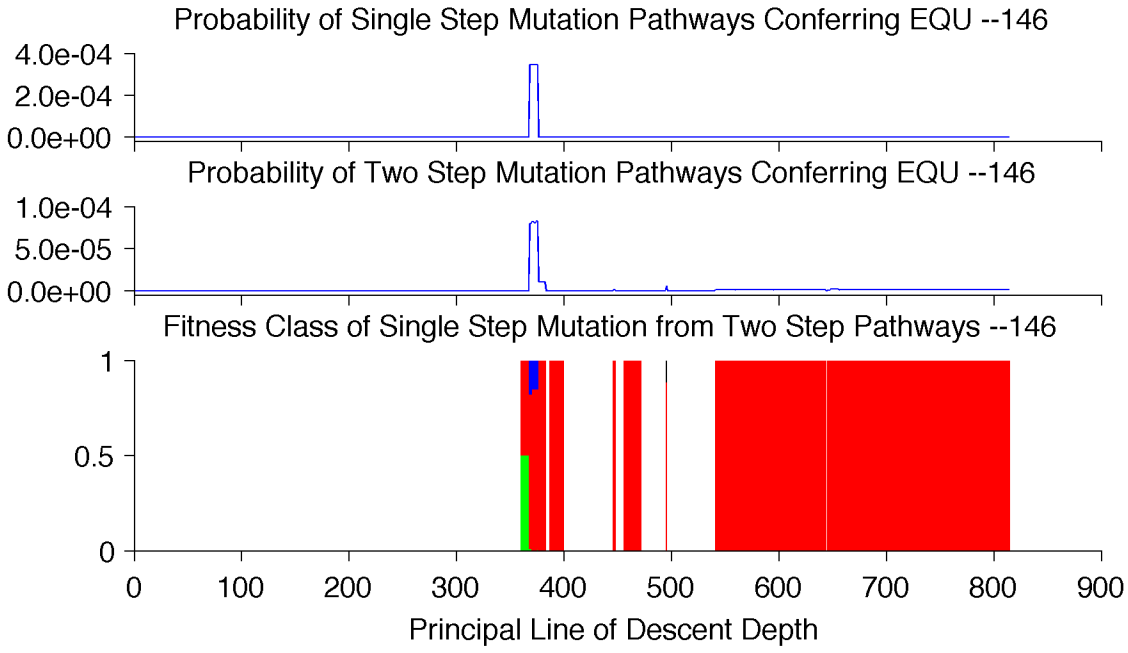

Figure A.46: Mutational neighborhood of the PLoD from population 146
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

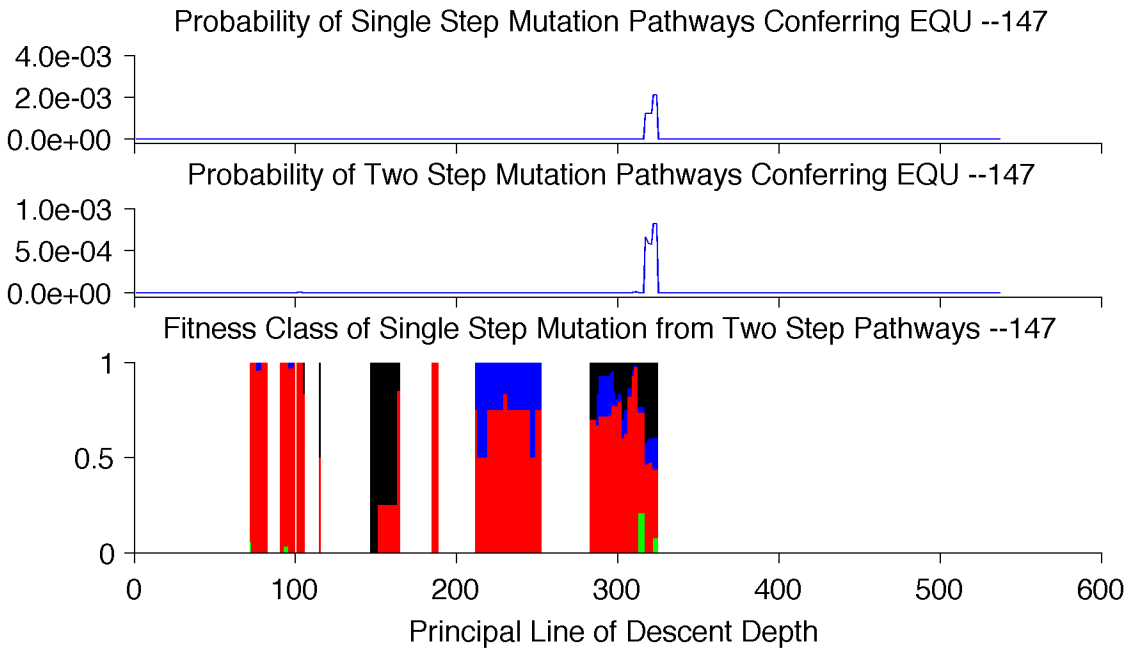

Figure A.47: Mutational neighborhood of the PLoD from population 147
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

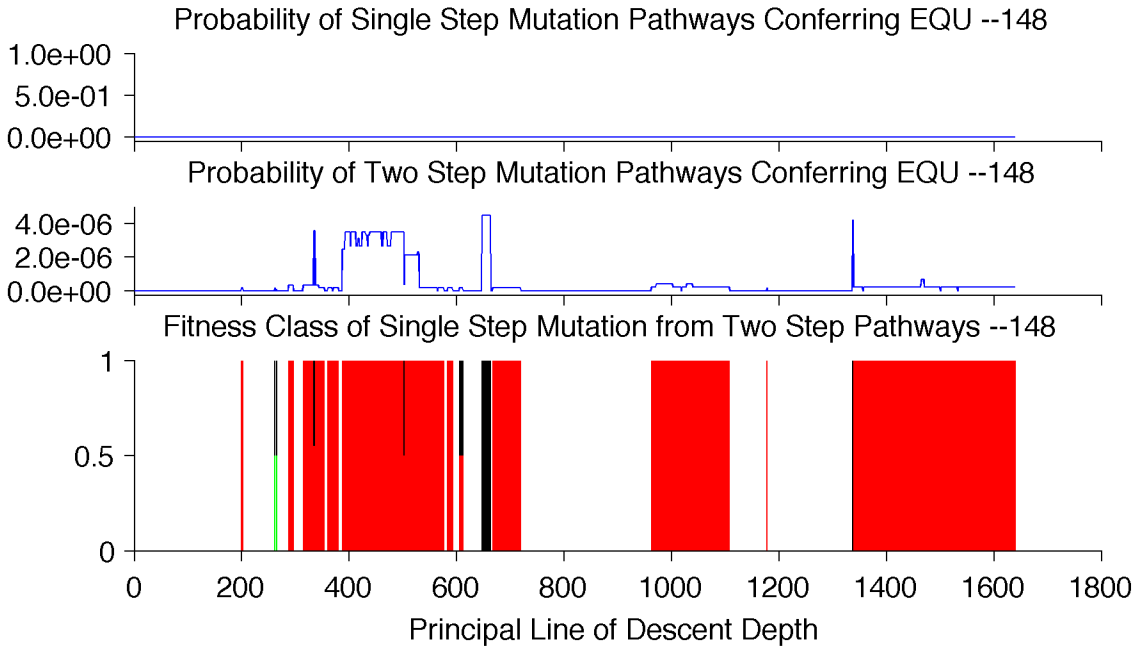

Figure A.48: Mutational neighborhood of the PLoD from population 148
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

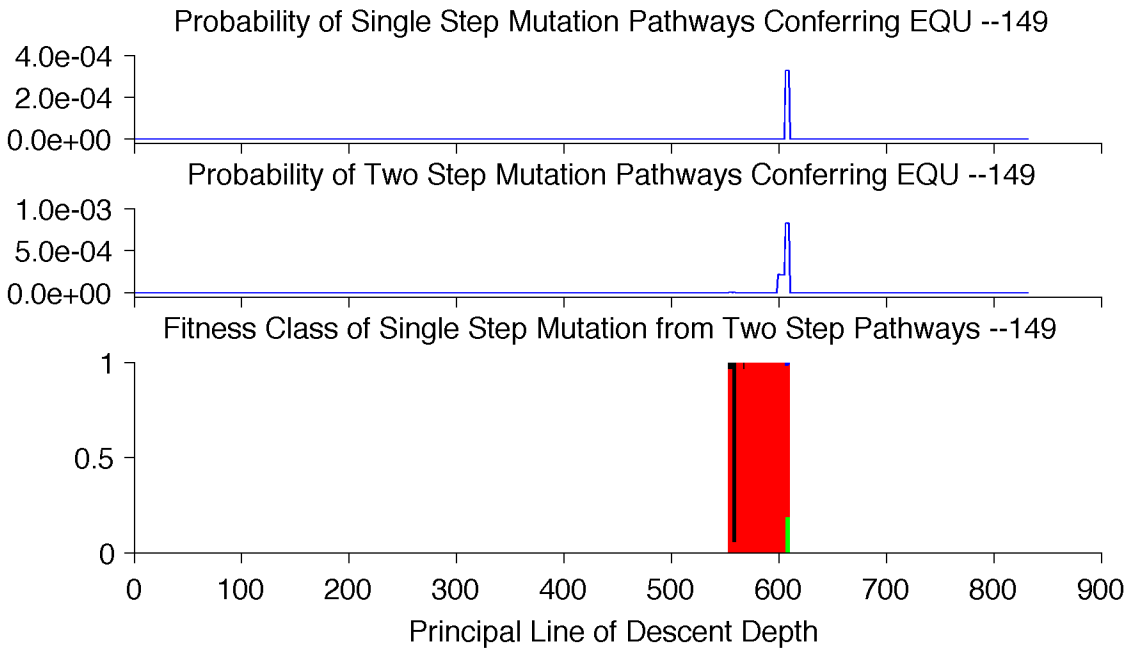

Figure A.49: Mutational neighborhood of the PLoD from population 149
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

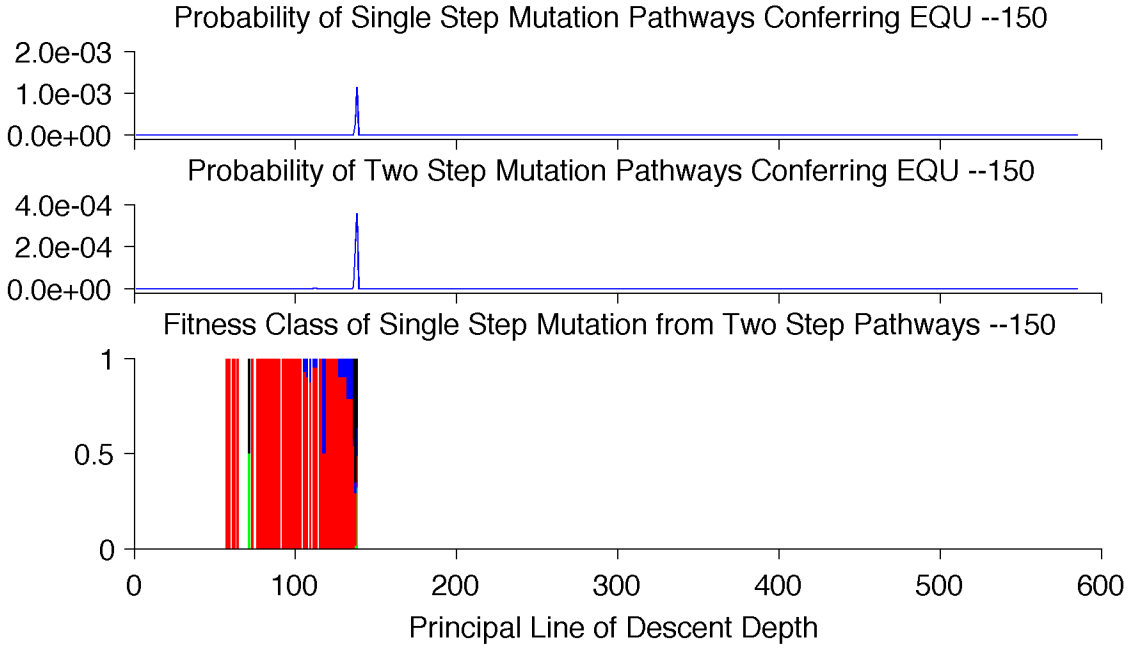

Figure A.50: Mutational neighborhood of the PLoD from population 150
Fitness classes shown: beneficial (green), neutral (blue), deleterious (red), lethal (black).

Appendix B

Complete History STEP Sampling Results

The STEP sampling results for all lineages analyzed in Section 3.2.3 are presented here.

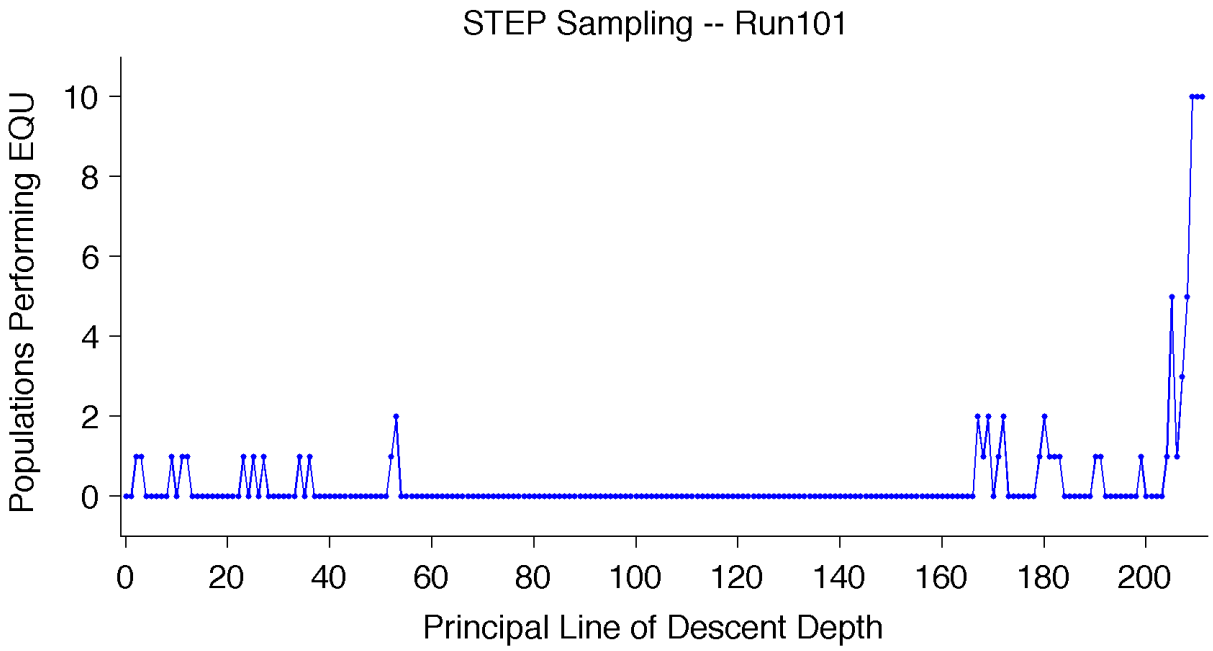

Figure B.1: STEP sampling along the PLoD from population 101.

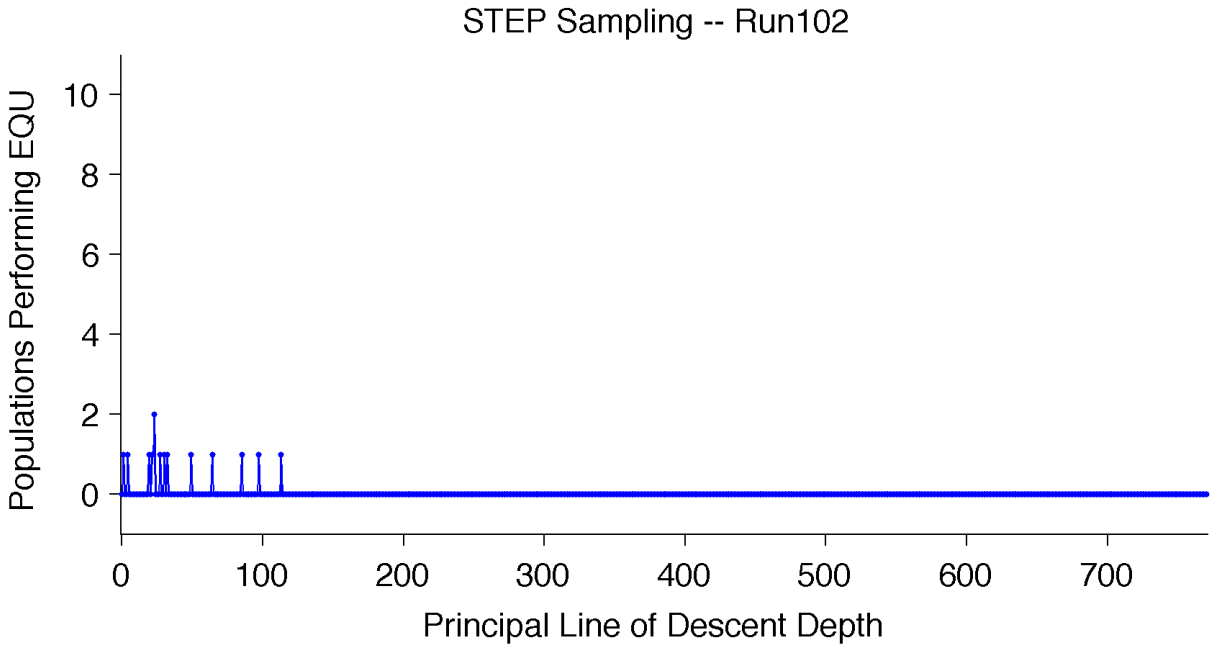

Figure B.2: STEP sampling along the PLoD from population 102.

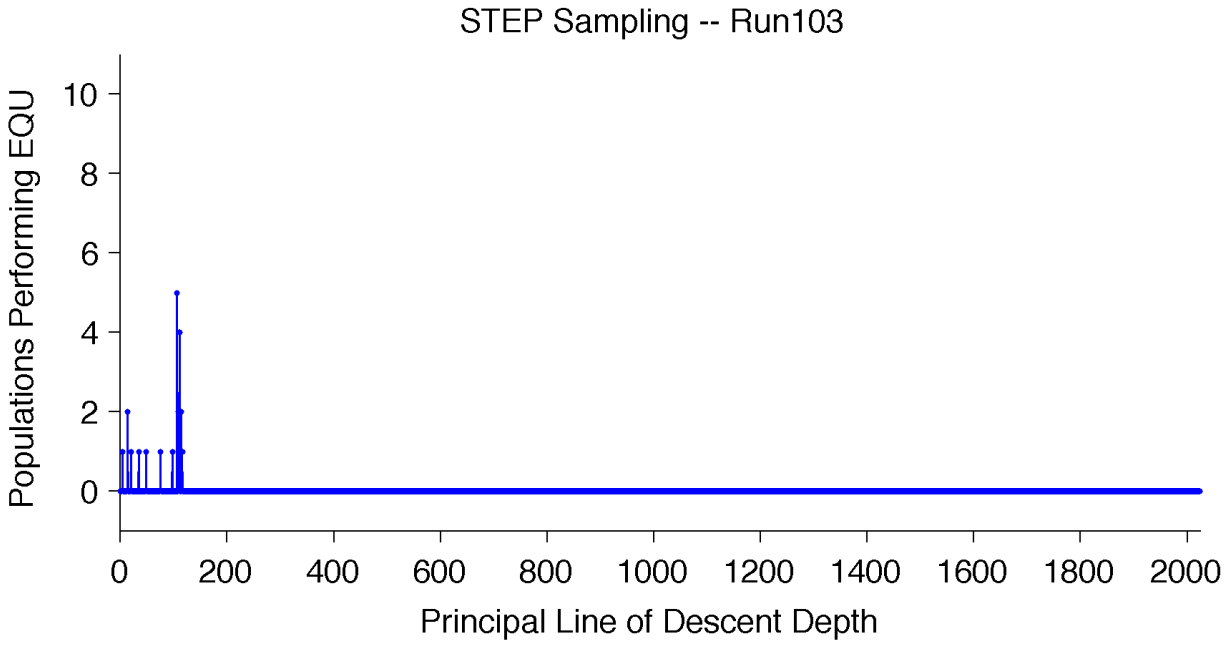

Figure B.3: STEP sampling along the PLoD from population 103.

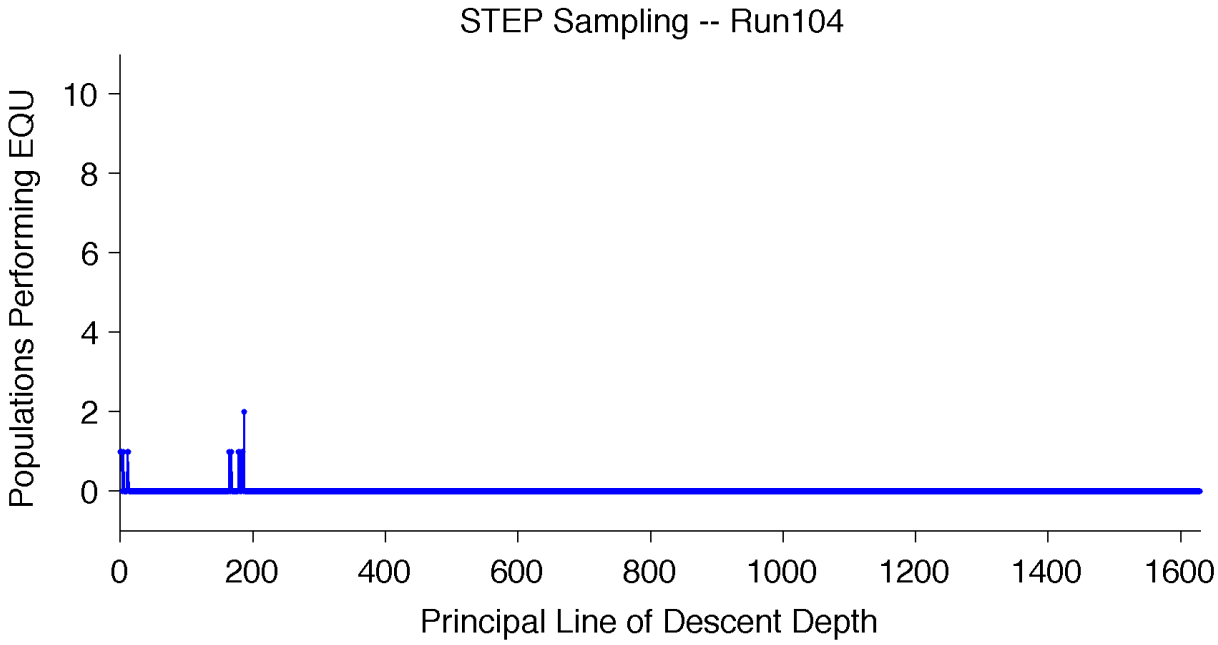

Figure B.4: STEP sampling along the PLoD from population 104.

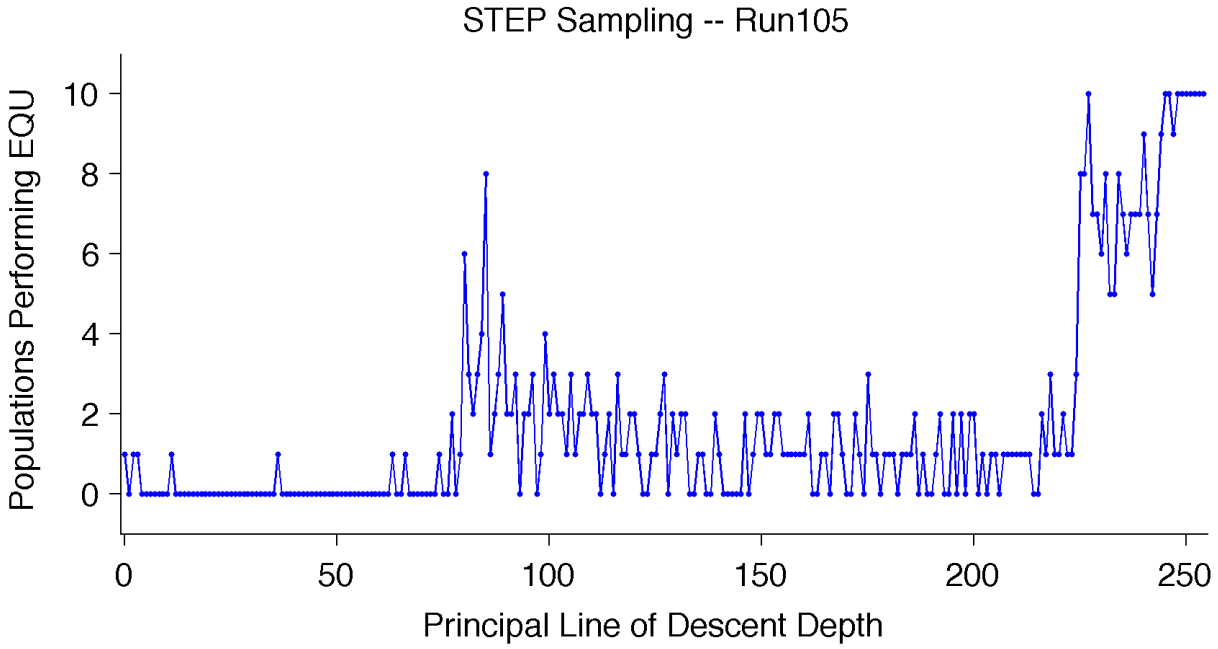

Figure B.5: STEP sampling along the PLoD from population 105.

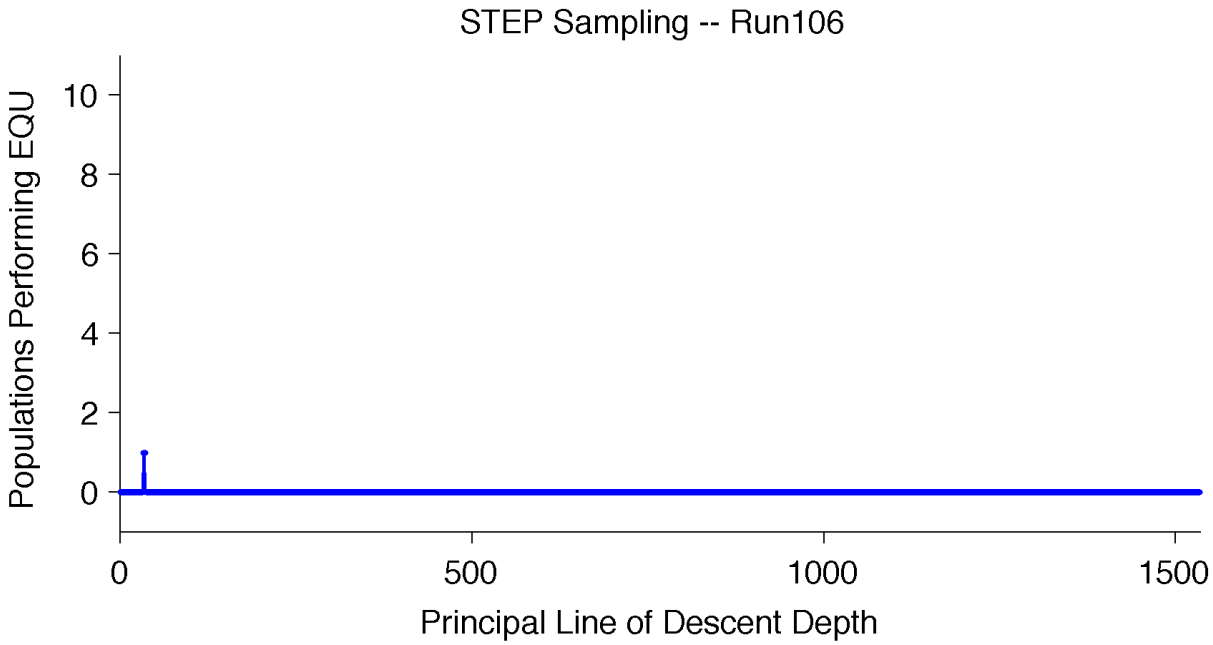

Figure B.6: STEP sampling along the PLoD from population 106.

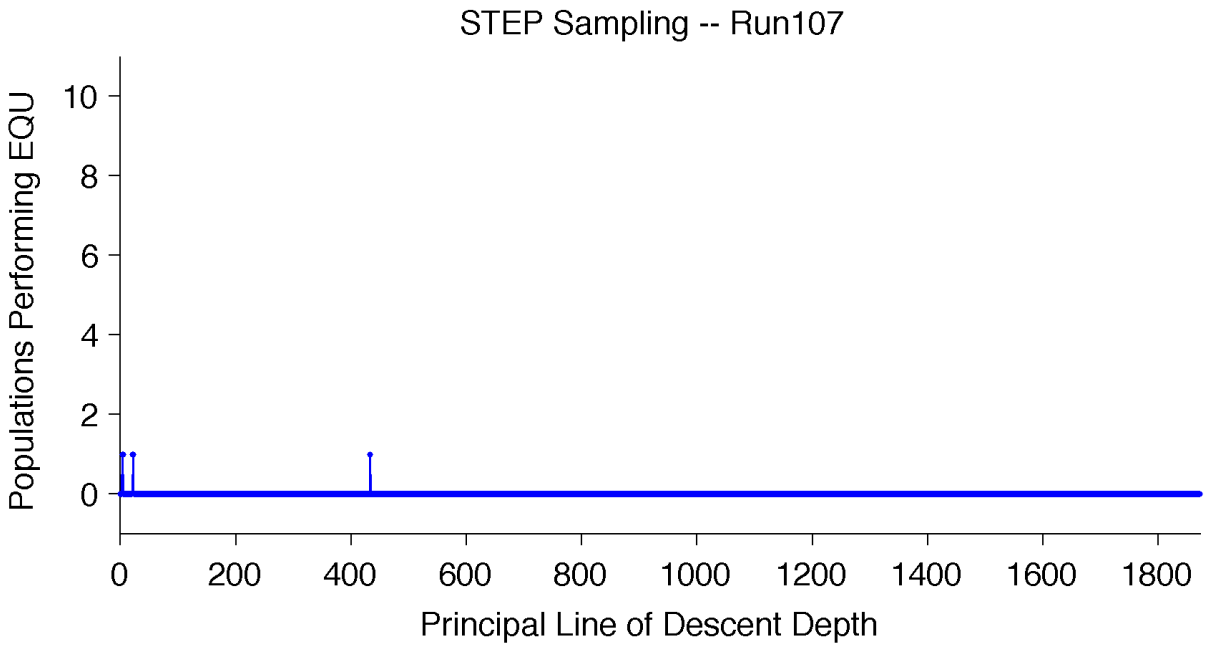

Figure B.7: STEP sampling along the PLoD from population 107.

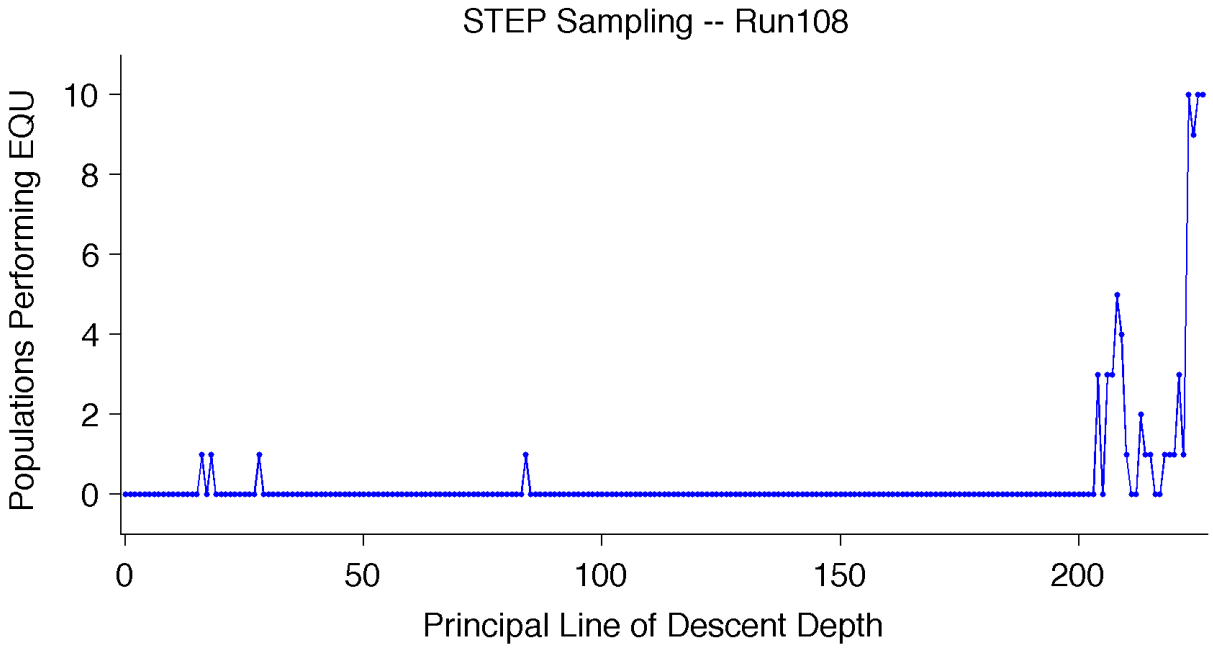

Figure B.8: STEP sampling along the PLoD from population 108.

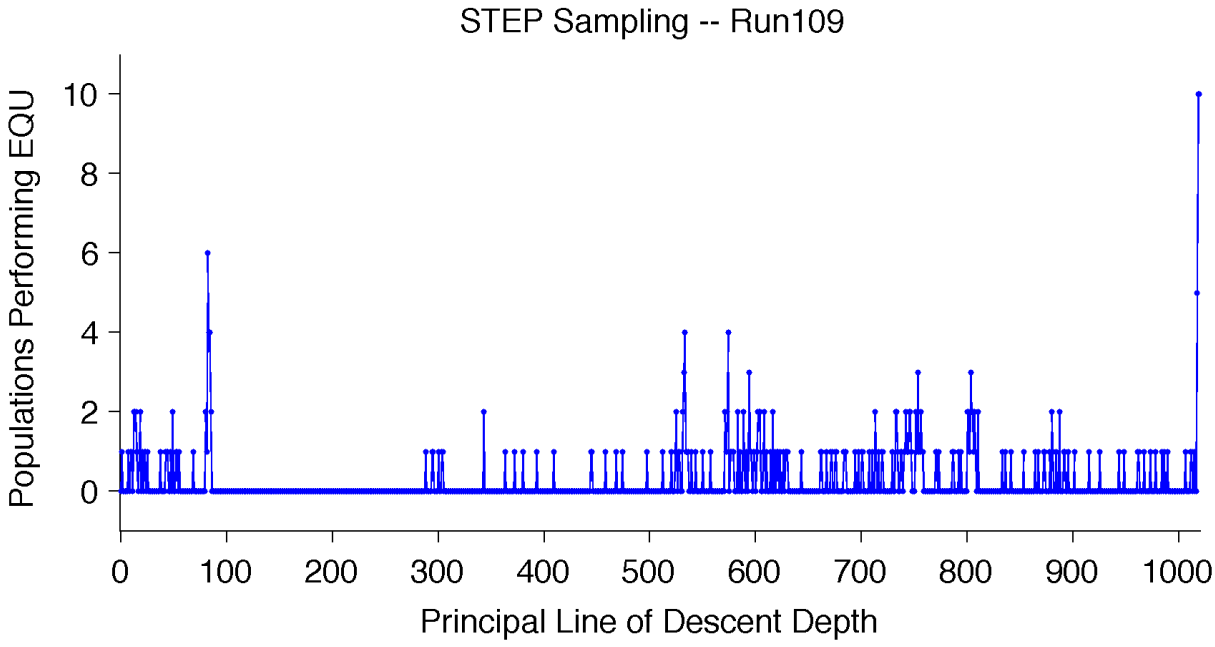

Figure B.9: STEP sampling along the PLoD from population 109.

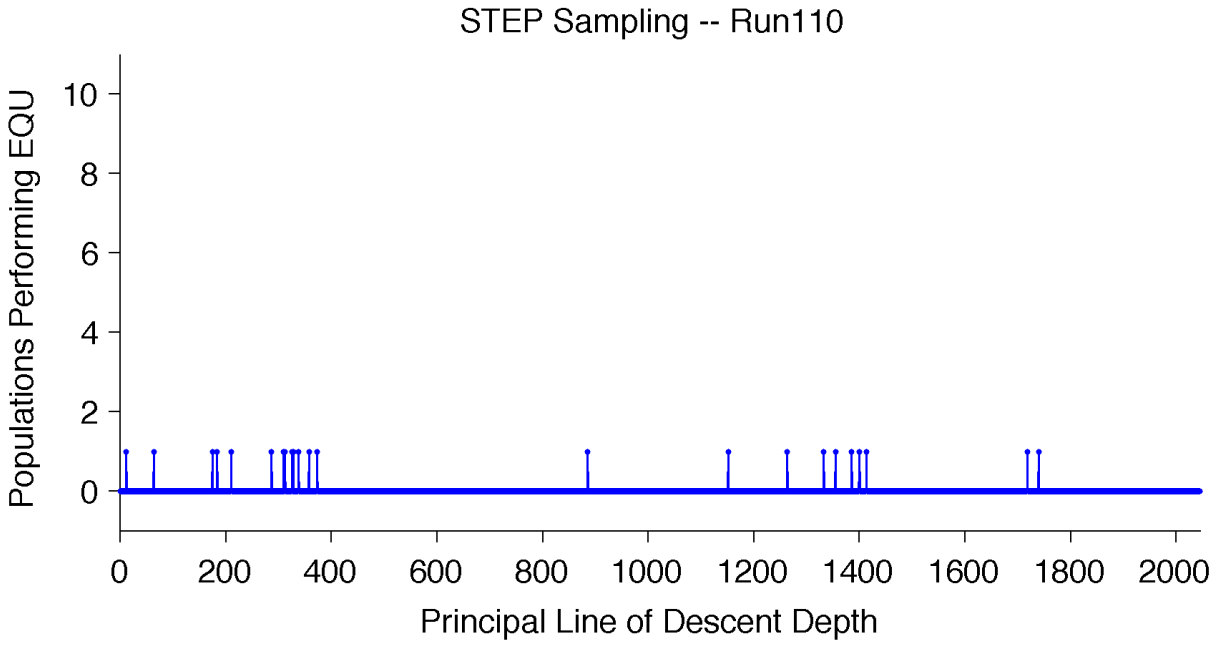

Figure B.10: STEP sampling along the PLoD from population 110.

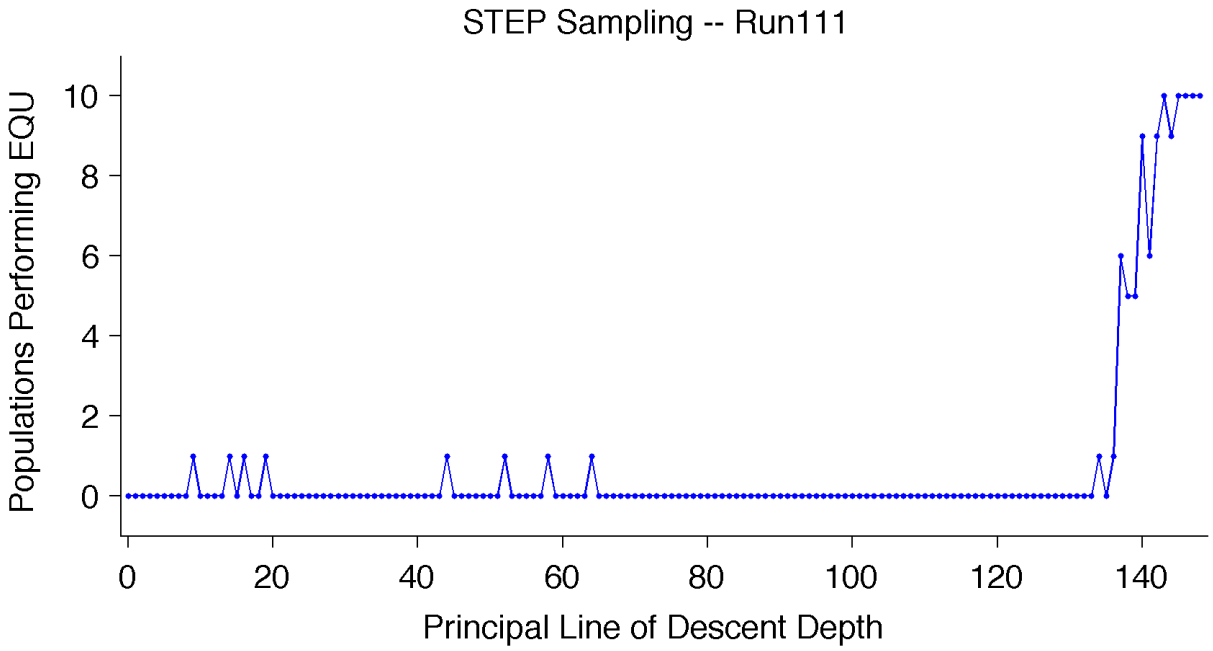

Figure B.11: STEP sampling along the PLoD from population 111.

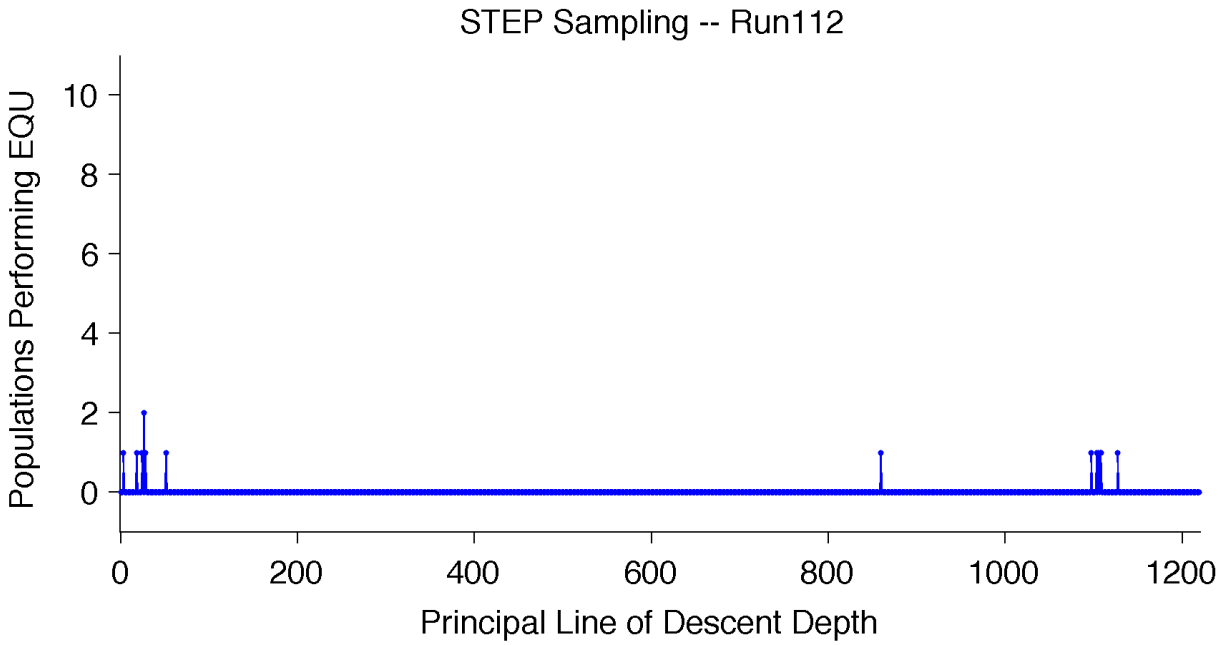

Figure B.12: STEP sampling along the PLoD from population 112.

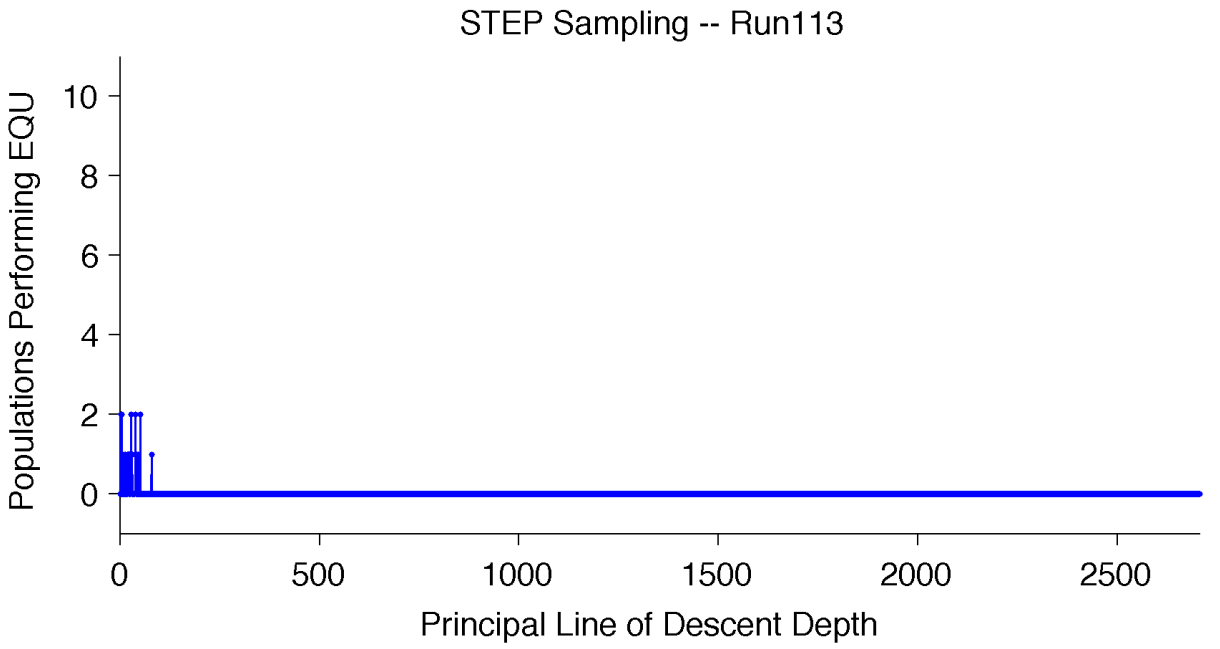

Figure B.13: STEP sampling along the PLoD from population 113.

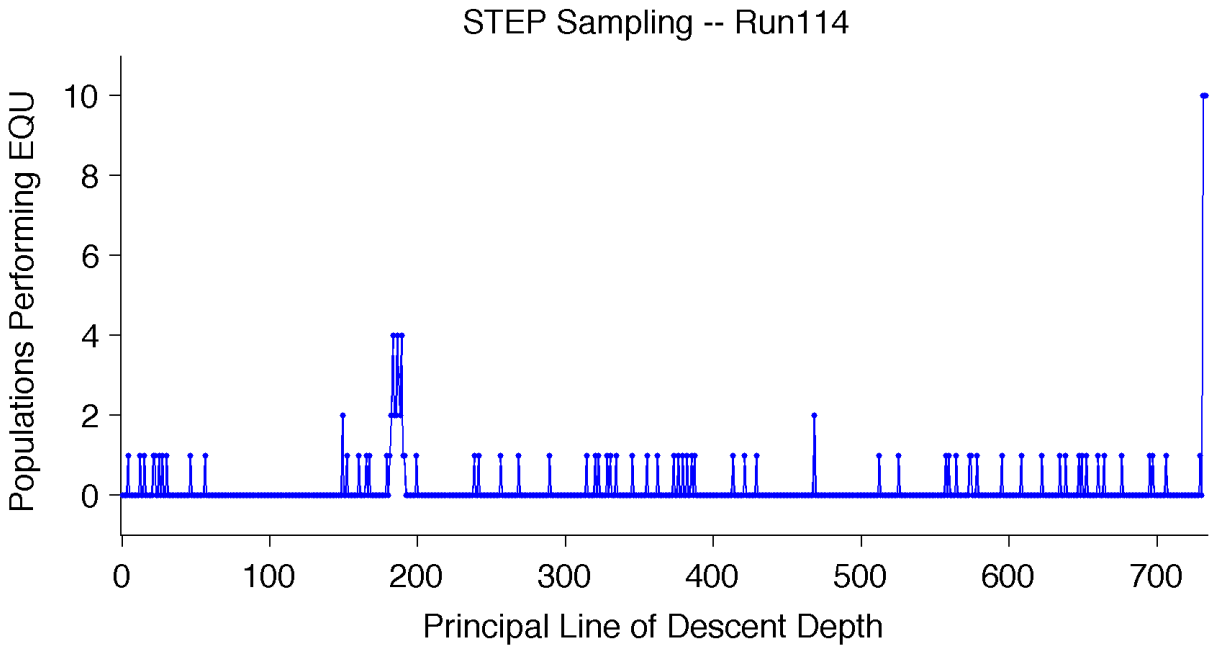

Figure B.14: STEP sampling along the PLoD from population 114.

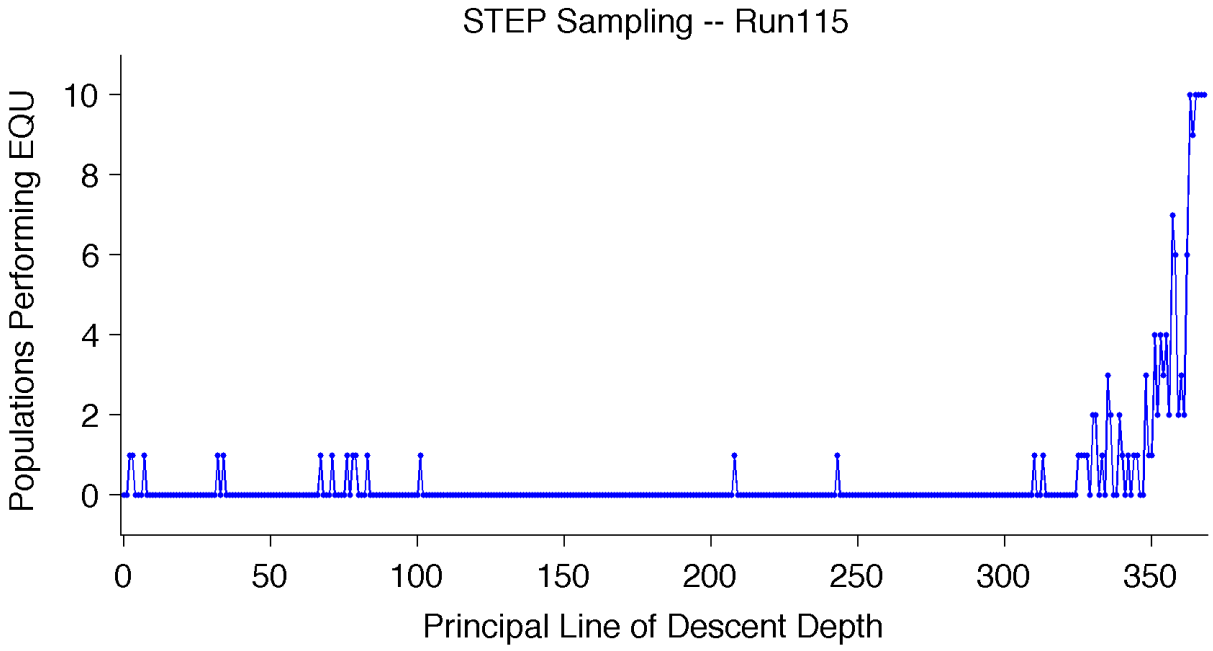

Figure B.15: STEP sampling along the PLoD from population 115.

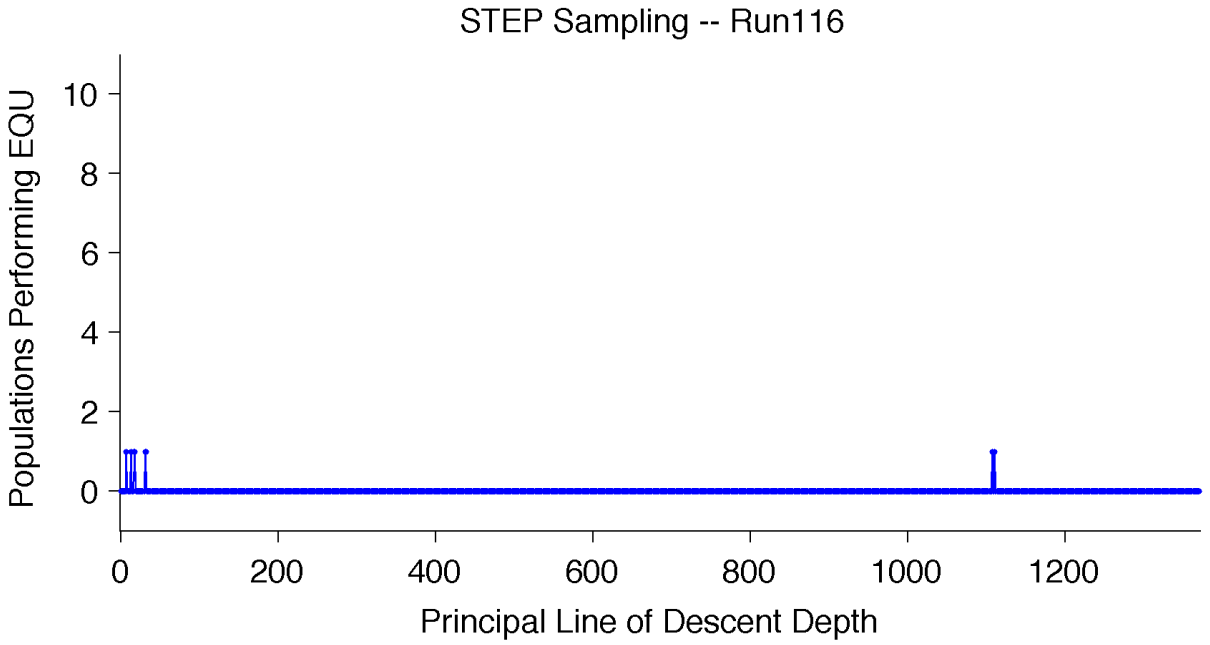

Figure B.16: STEP sampling along the PLoD from population 116.

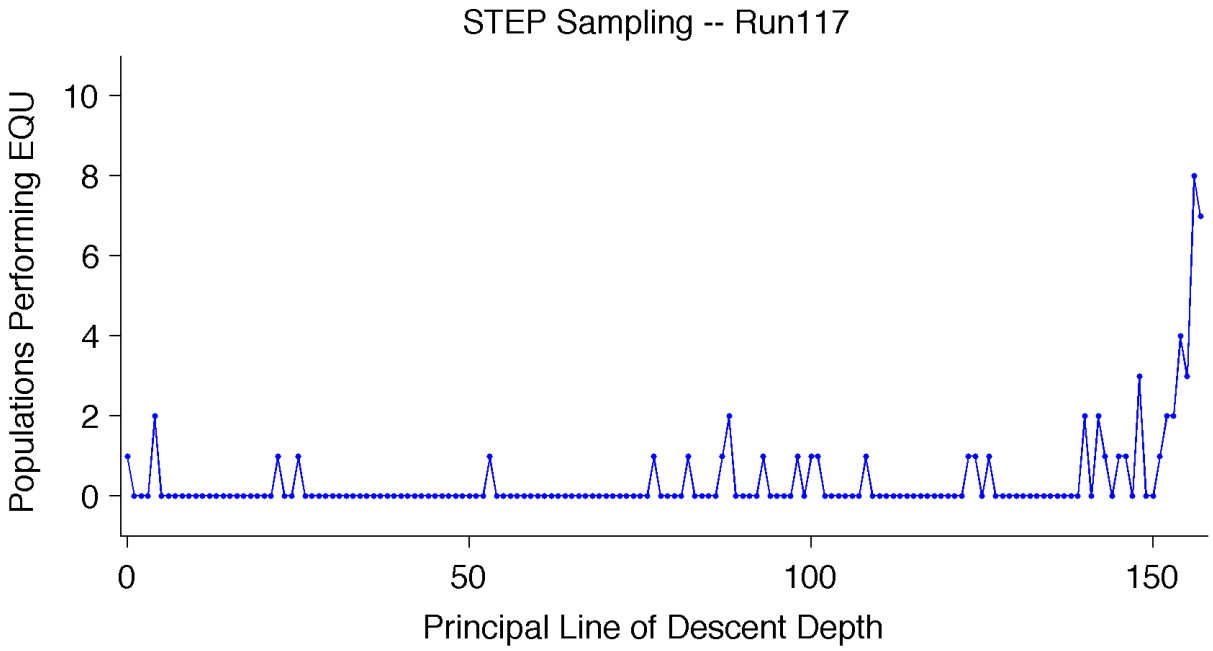

Figure B.17: STEP sampling along the PLoD from population 117.

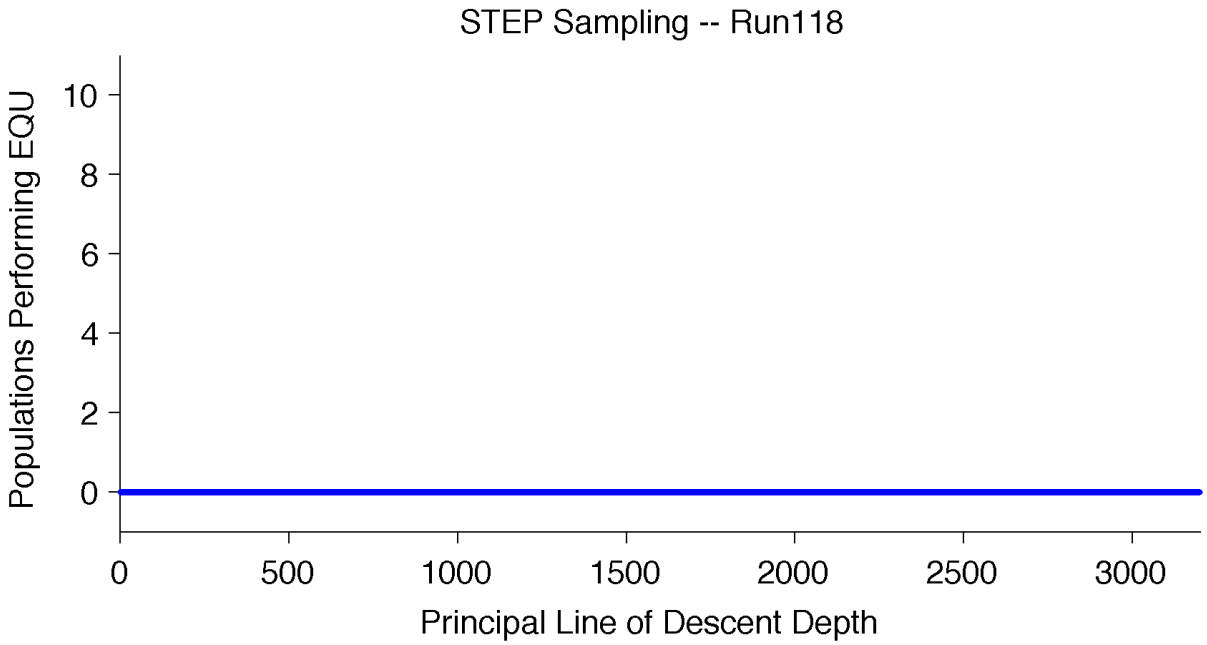

Figure B.18: STEP sampling along the PLoD from population 118.

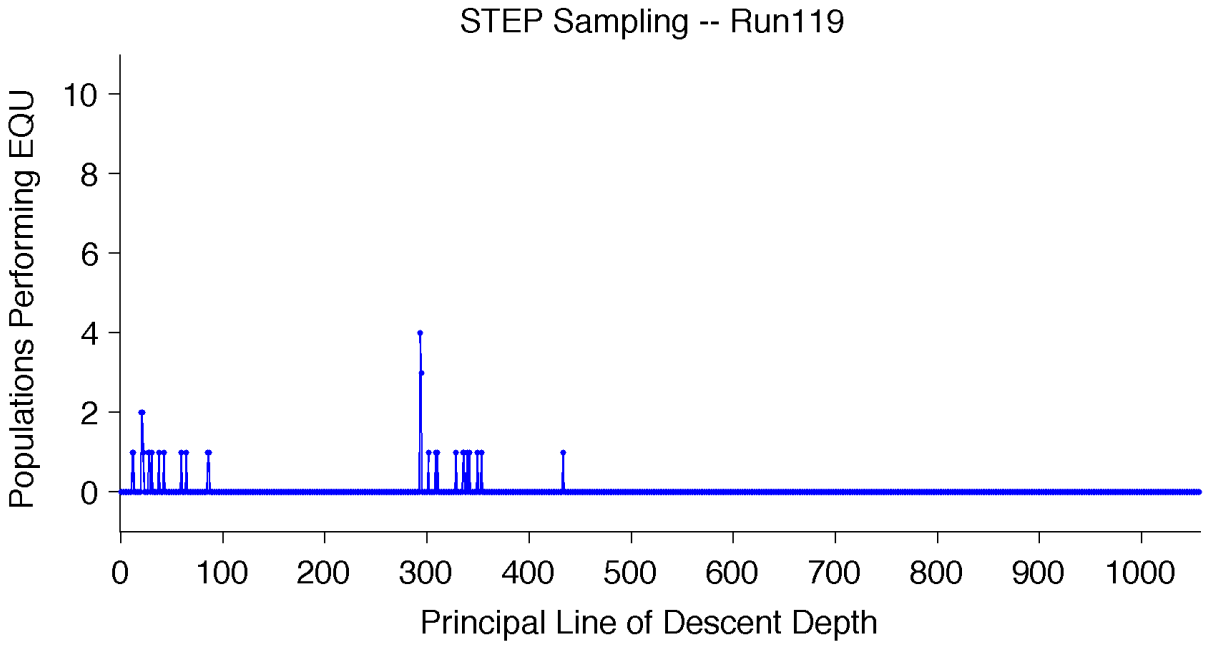

Figure B.19: STEP sampling along the PLoD from population 119.

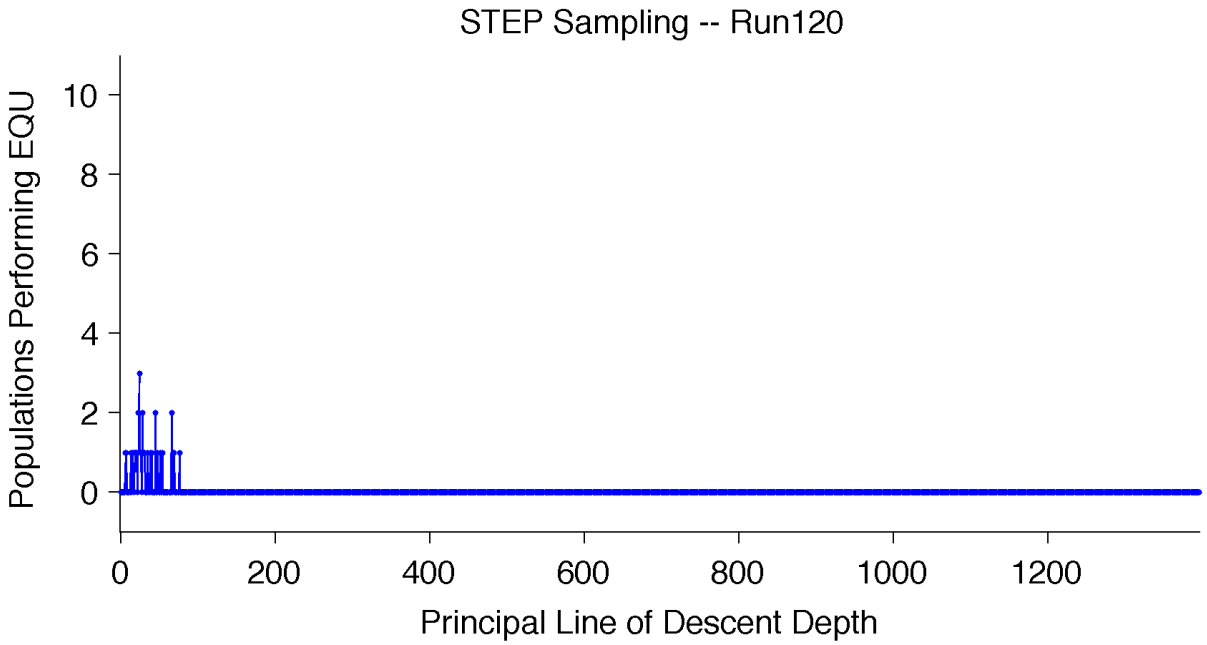

Figure B.20: STEP sampling along the PLoD from population 120.

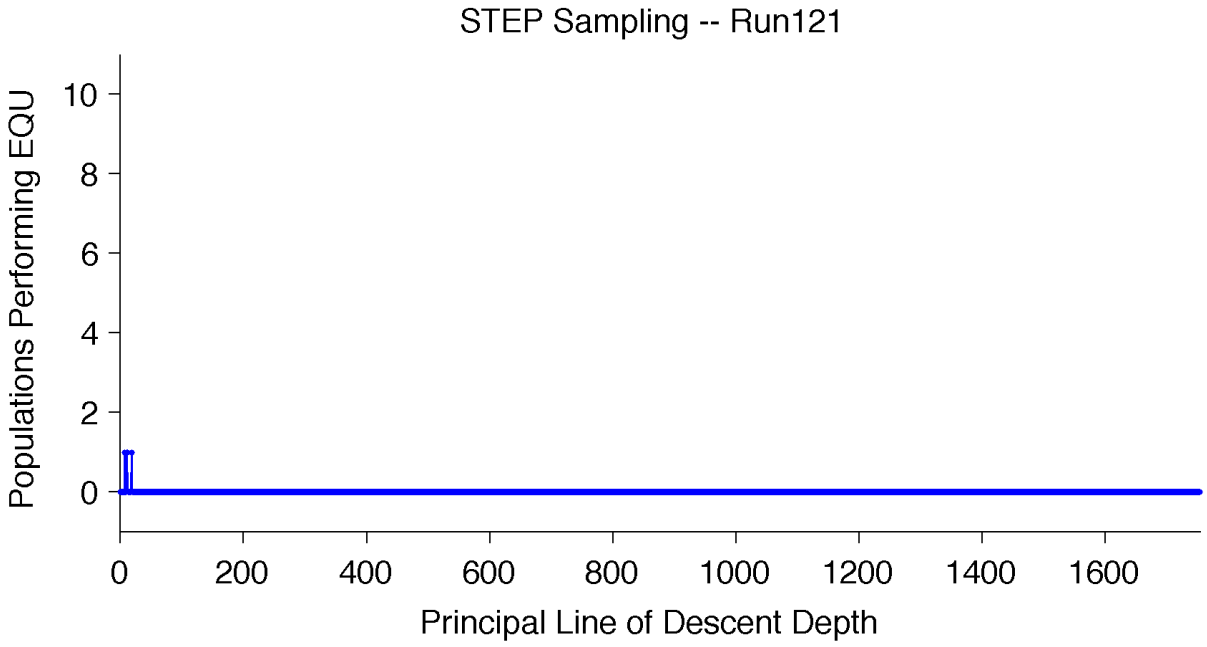

Figure B.21: STEP sampling along the PLoD from population 121.

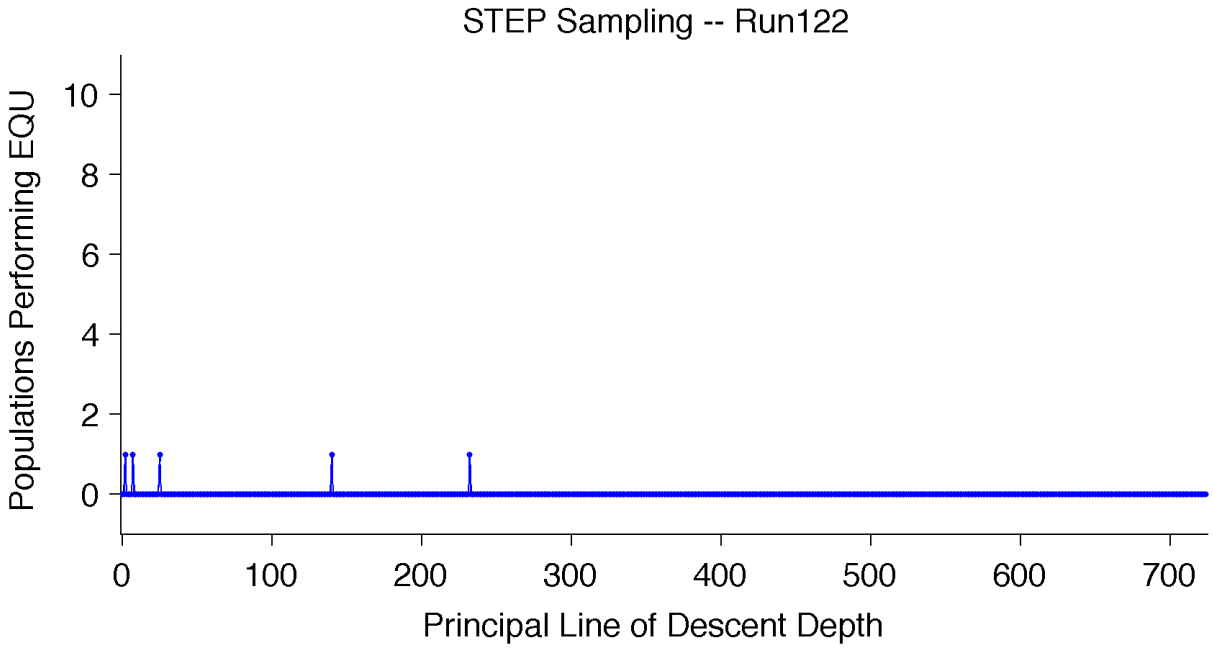

Figure B.22: STEP sampling along the PLoD from population 122.

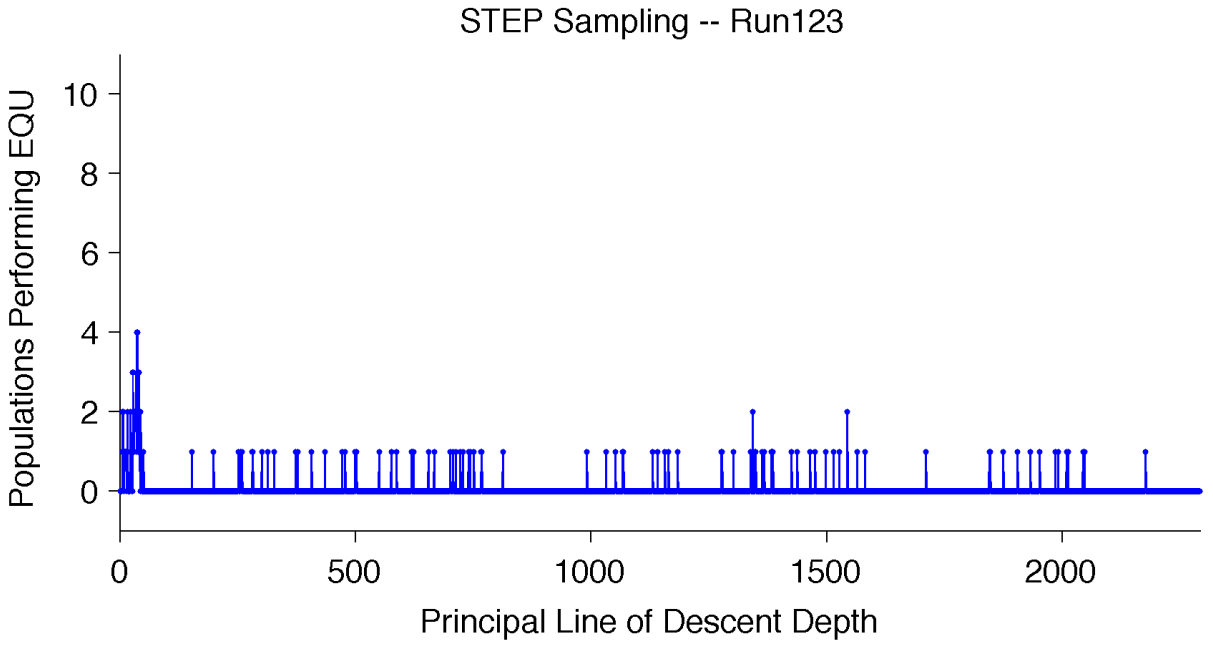

Figure B.23: STEP sampling along the PLoD from population 123.

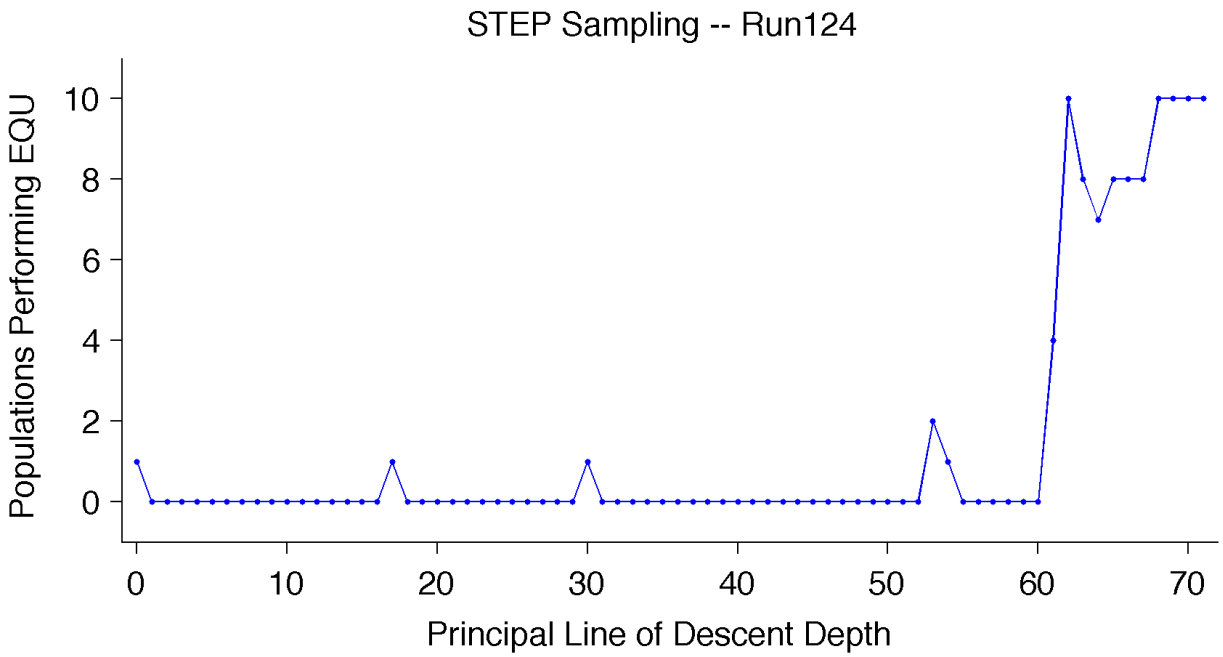

Figure B.24: STEP sampling along the PLoD from population 124.

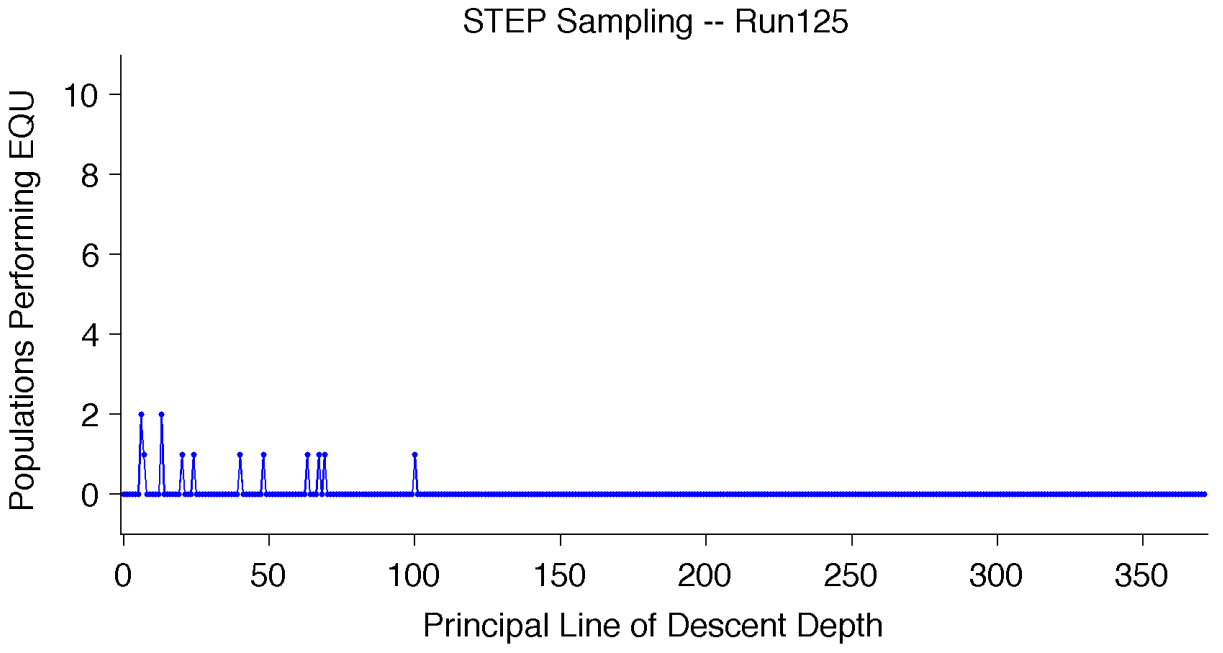

Figure B.25: STEP sampling along the PLoD from population 125.

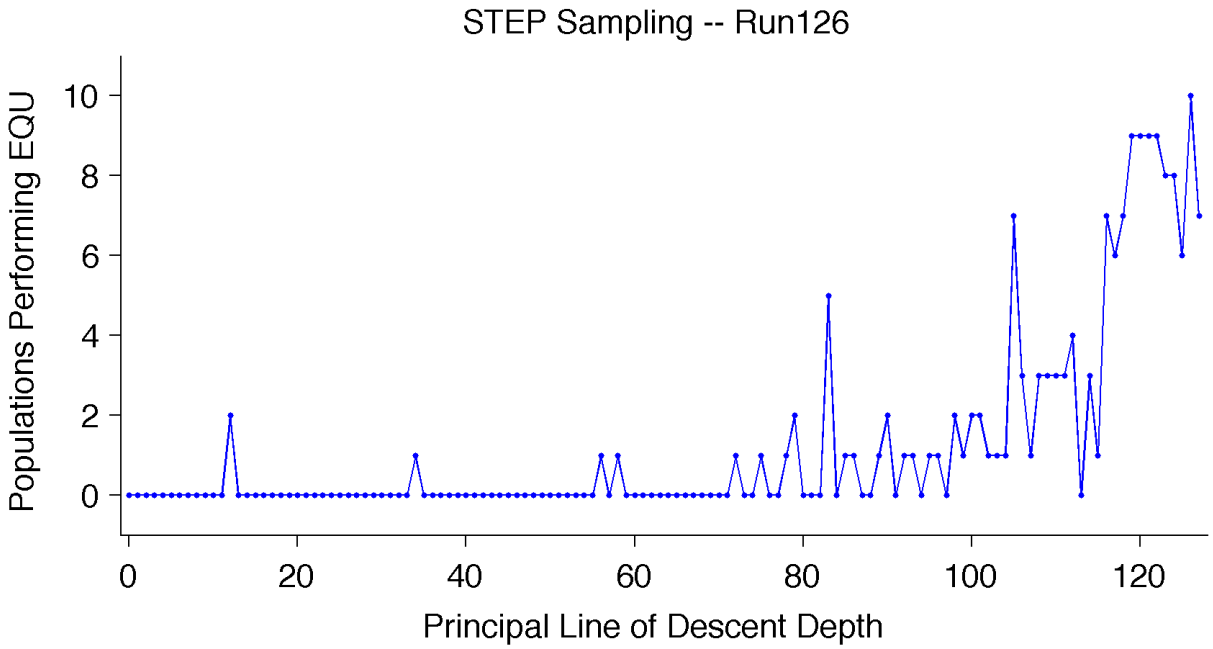

Figure B.26: STEP sampling along the PLoD from population 126.

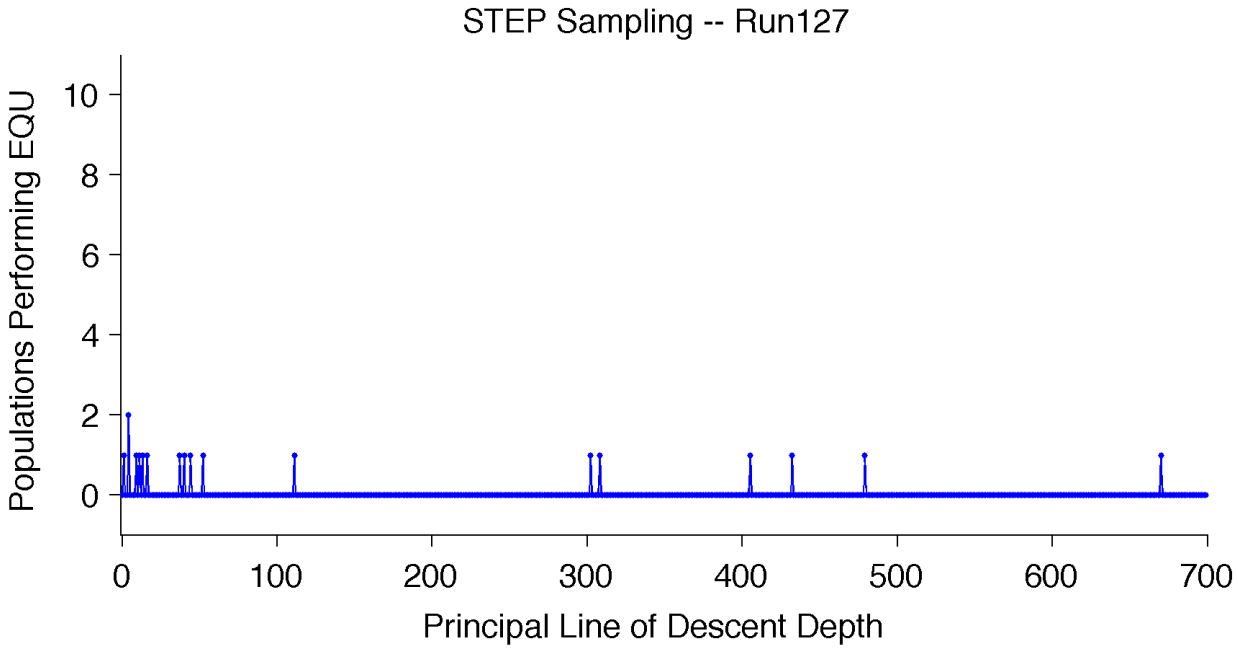

Figure B.27: STEP sampling along the PLoD from population 127.

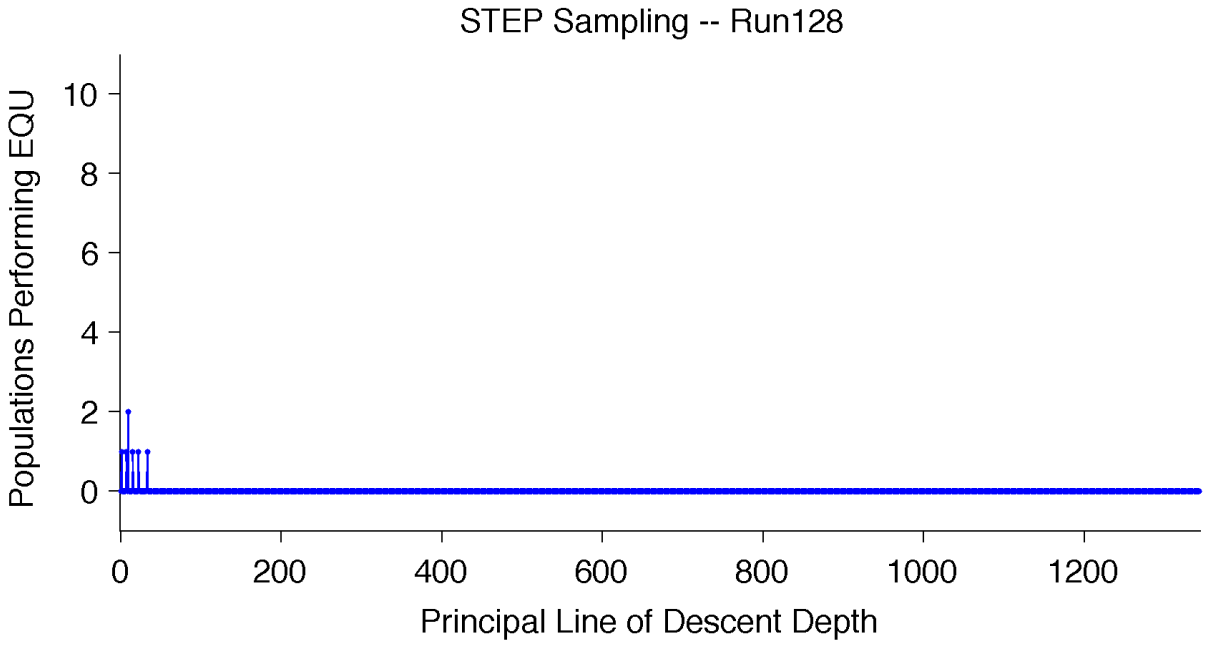

Figure B.28: STEP sampling along the PLoD from population 128.

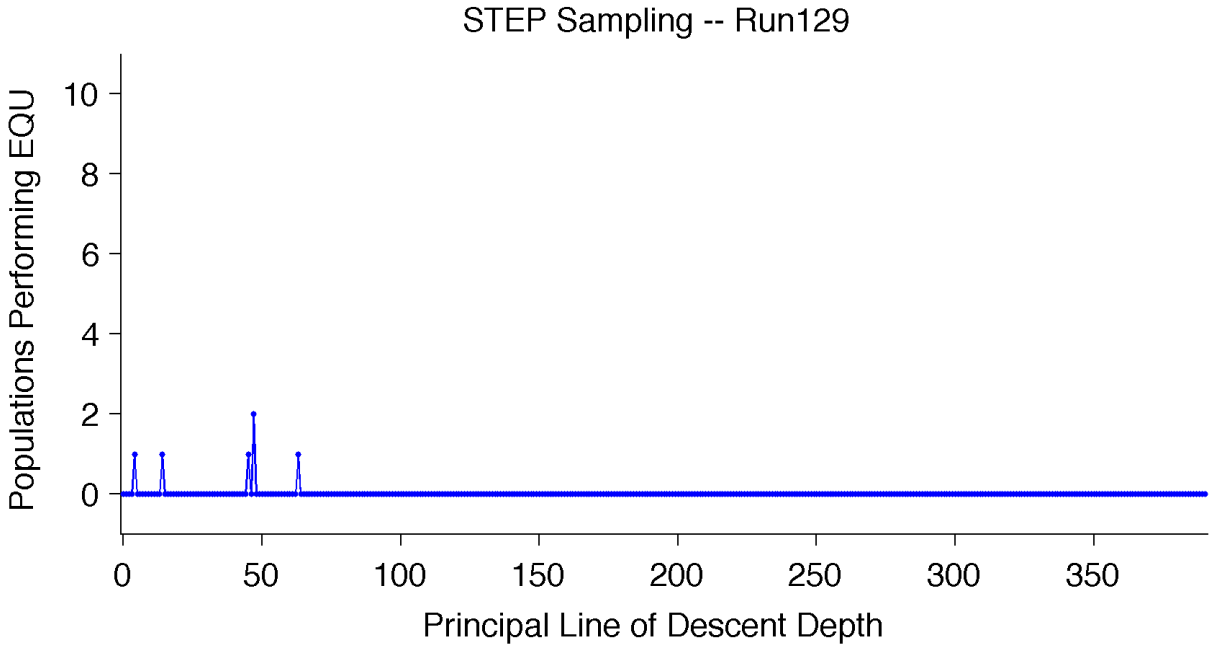

Figure B.29: STEP sampling along the PLoD from population 129.

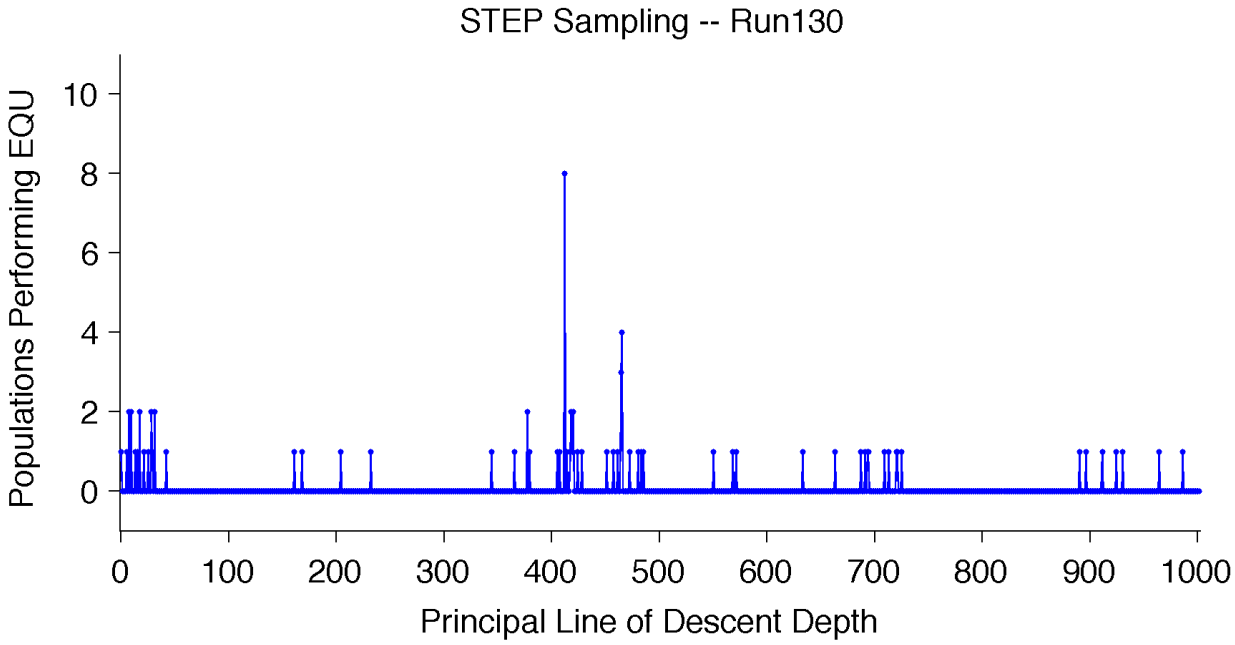

Figure B.30: STEP sampling along the PLoD from population 130.

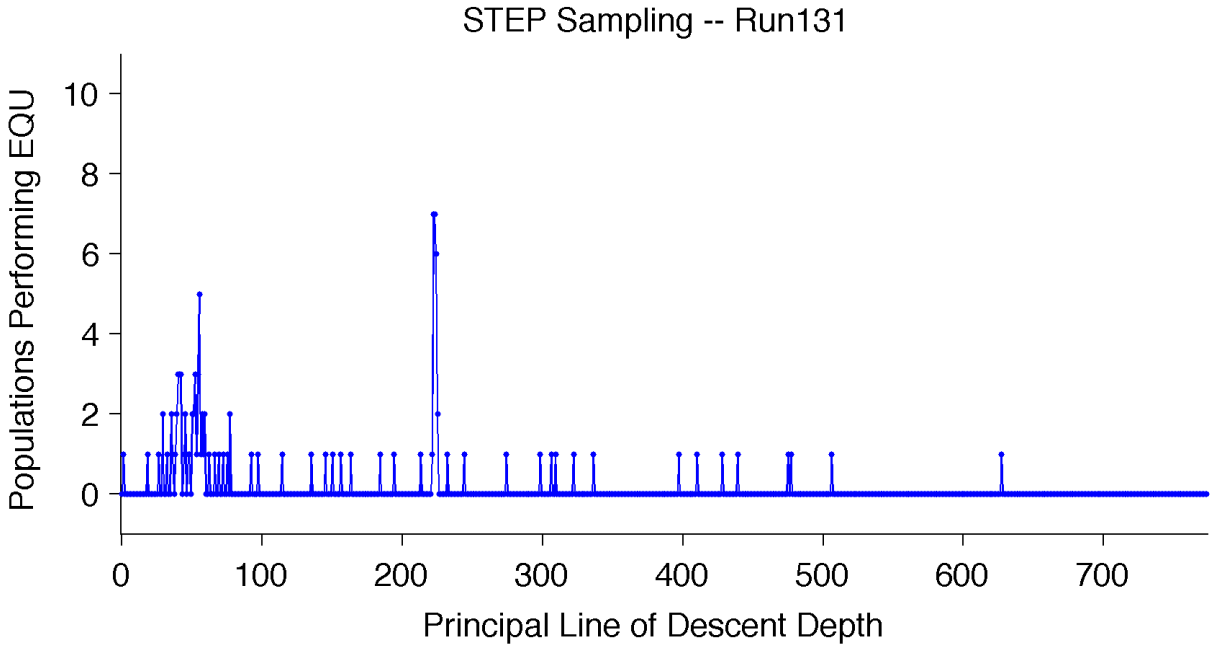

Figure B.31: STEP sampling along the PLoD from population 131.

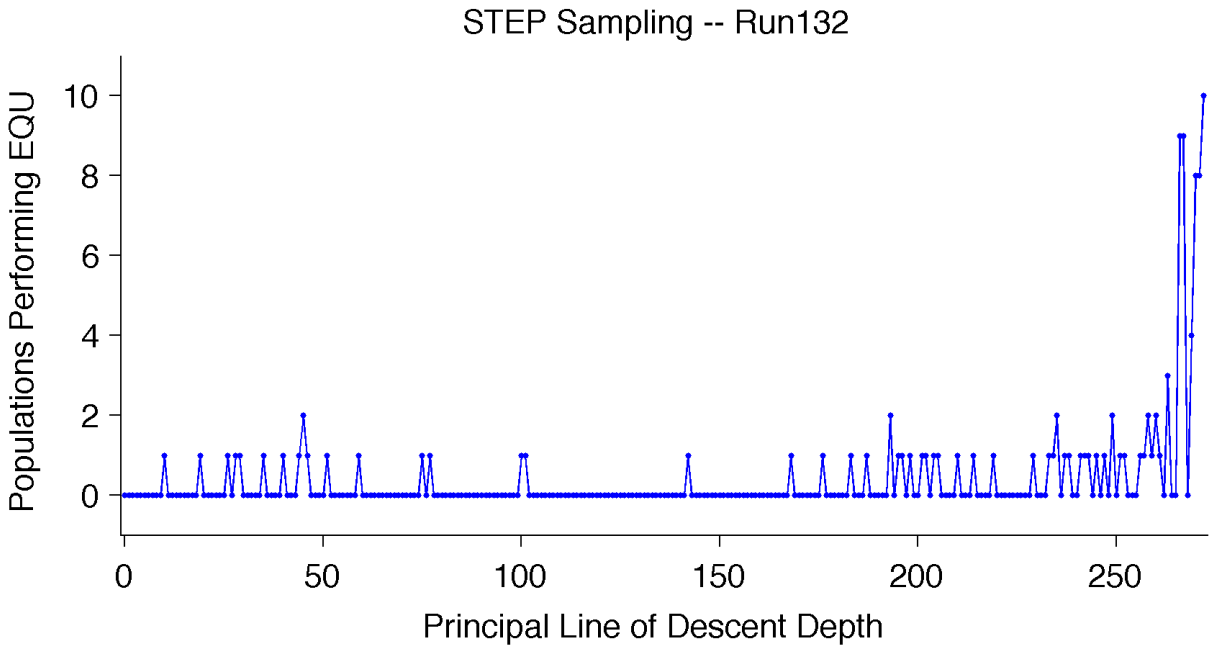

Figure B.32: STEP sampling along the PLoD from population 132.

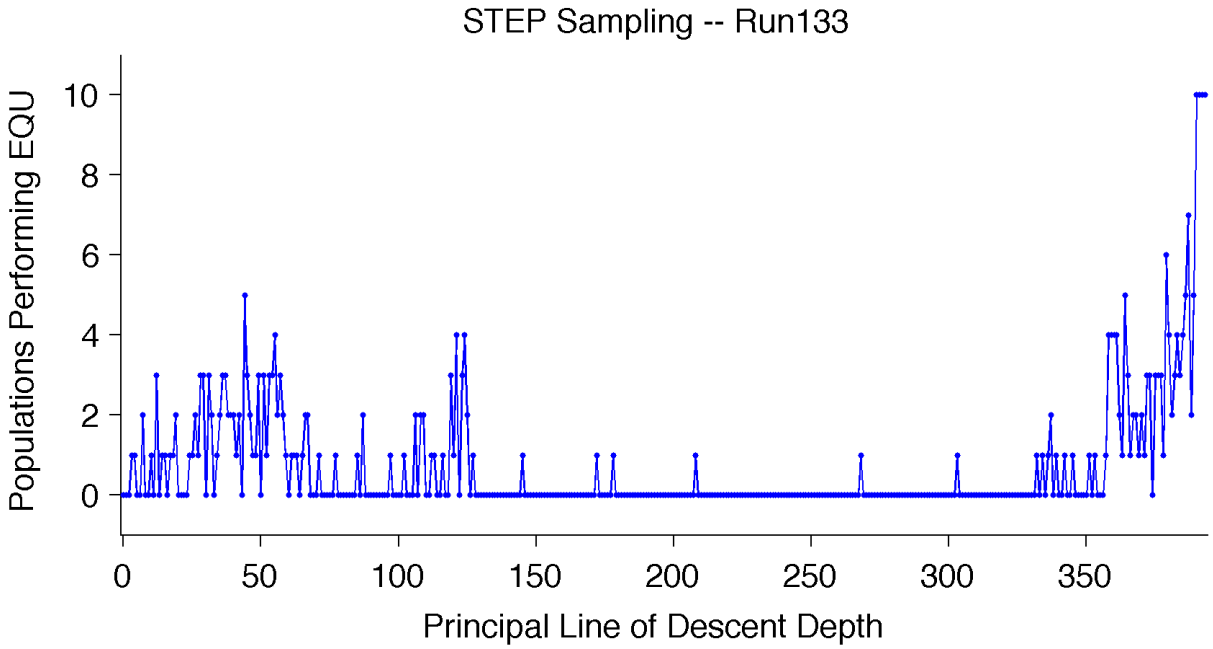

Figure B.33: STEP sampling along the PLoD from population 133.

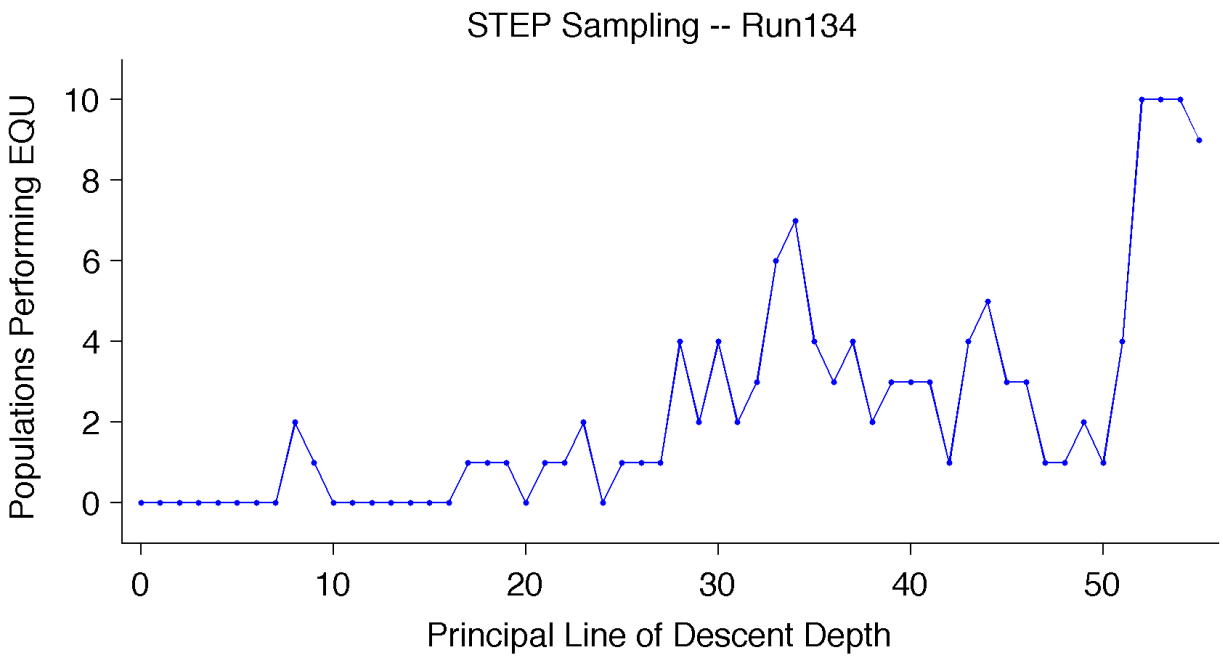

Figure B.34: STEP sampling along the PLoD from population 134.

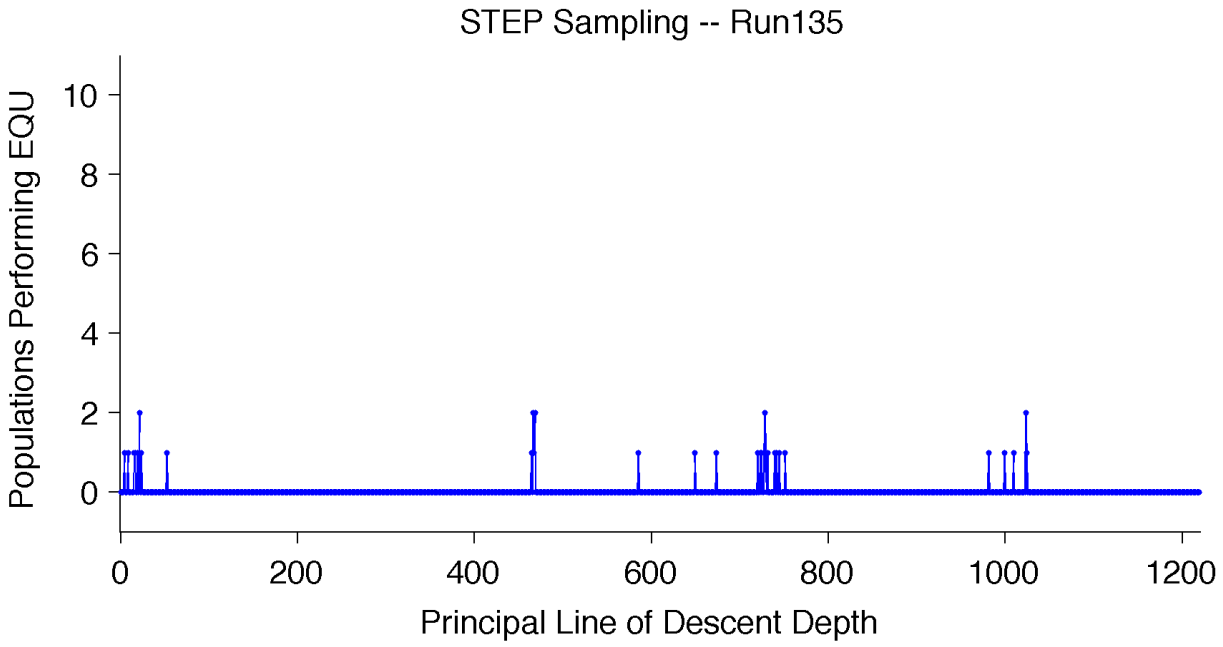

Figure B.35: STEP sampling along the PLoD from population 135.

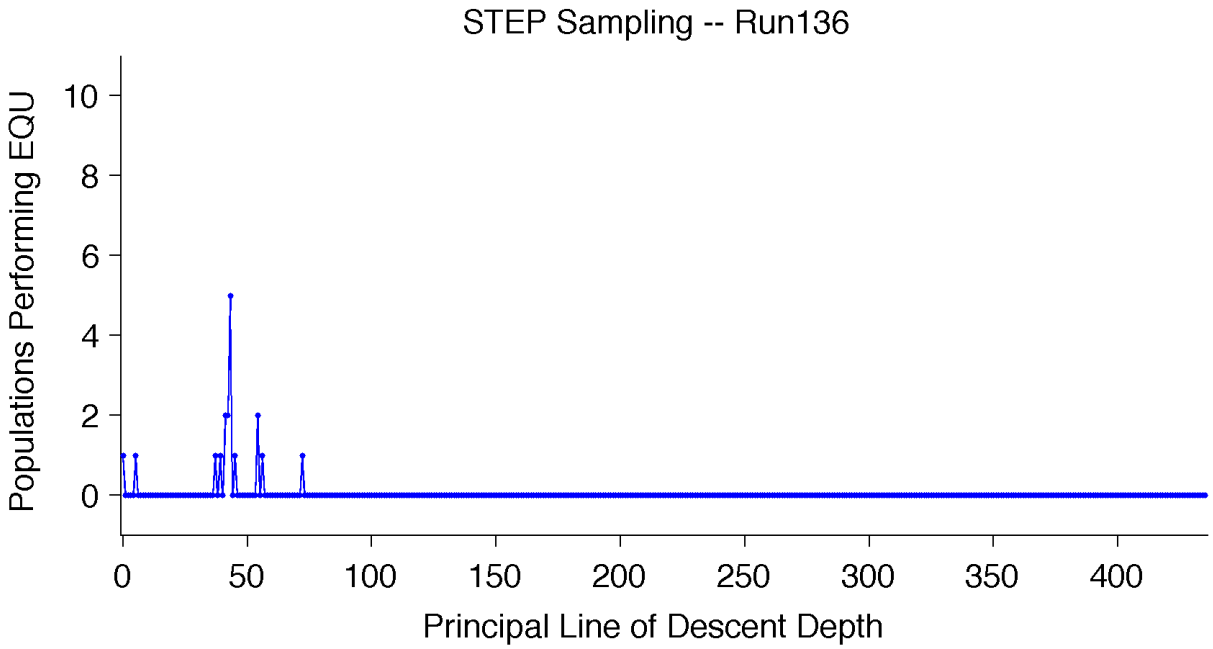

Figure B.36: STEP sampling along the PLoD from population 136.

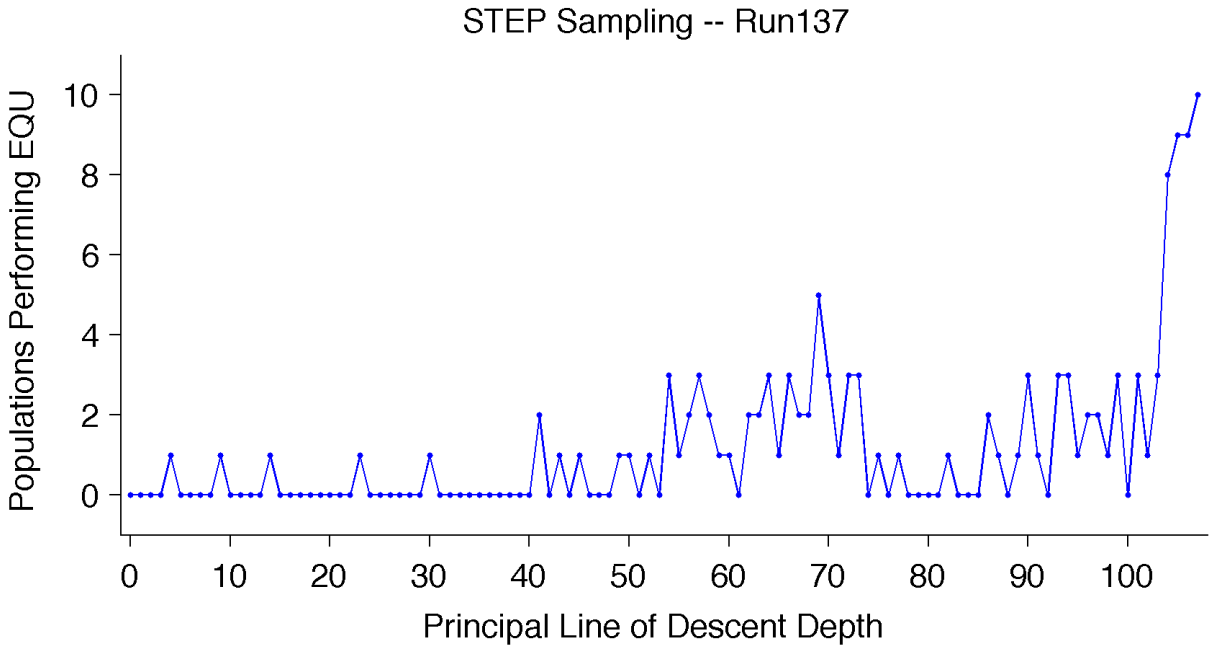

Figure B.37: STEP sampling along the PLoD from population 137.

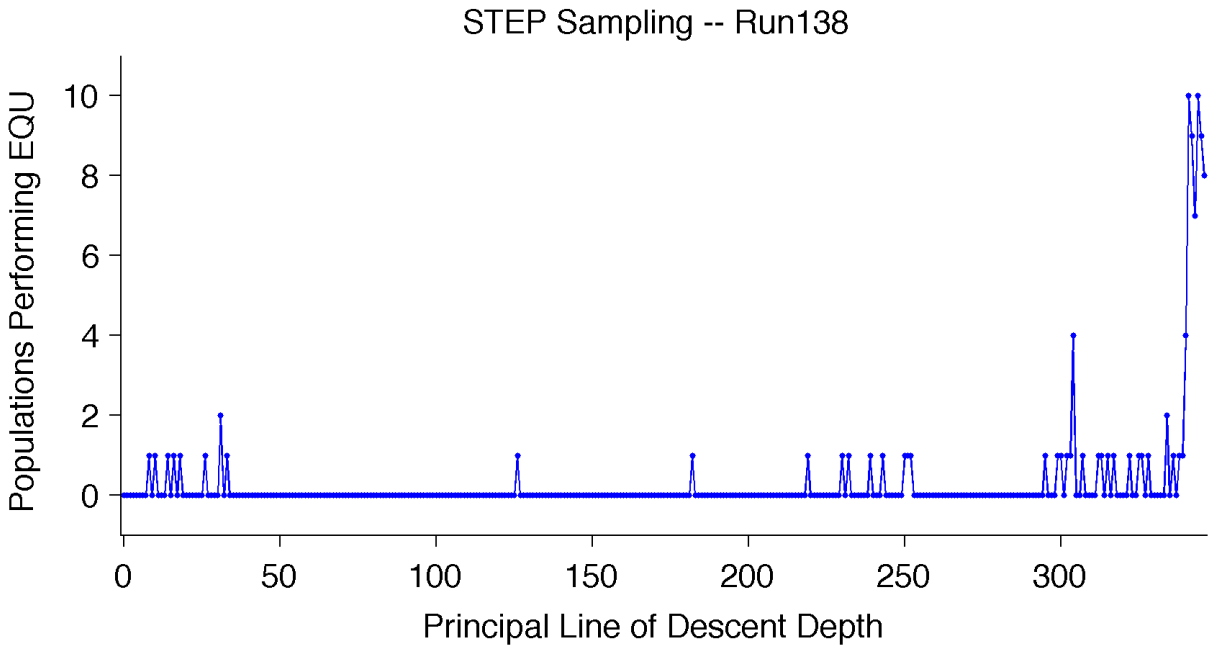

Figure B.38: STEP sampling along the PLoD from population 138.

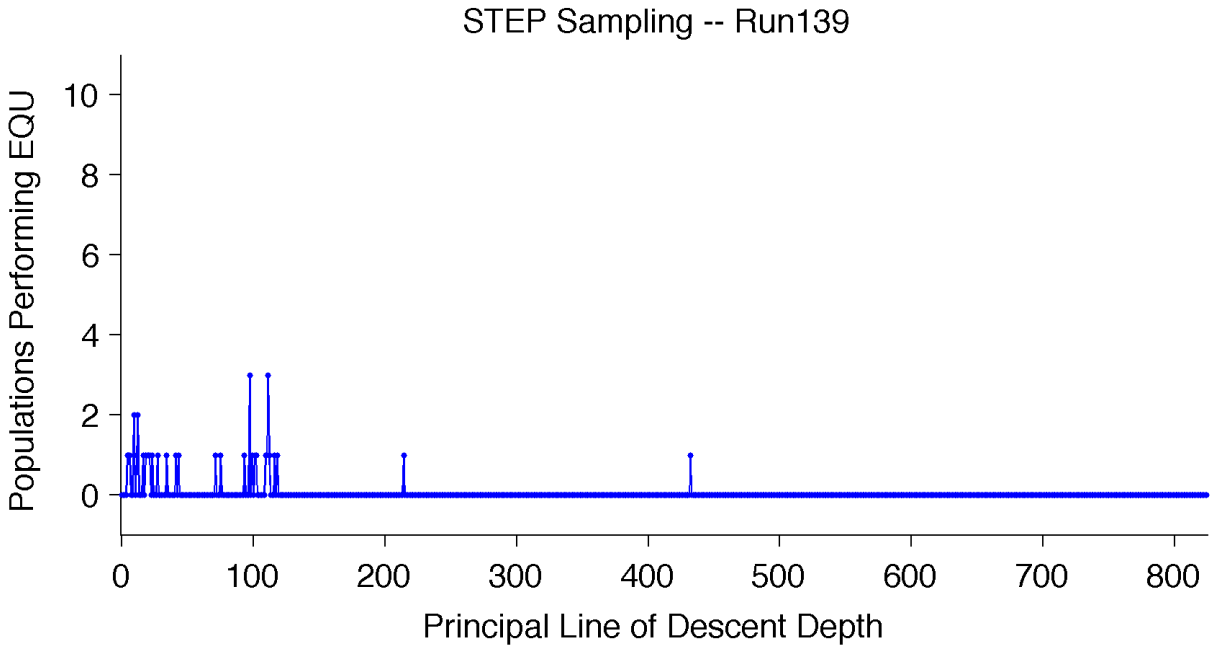

Figure B.39: STEP sampling along the PLoD from population 139.

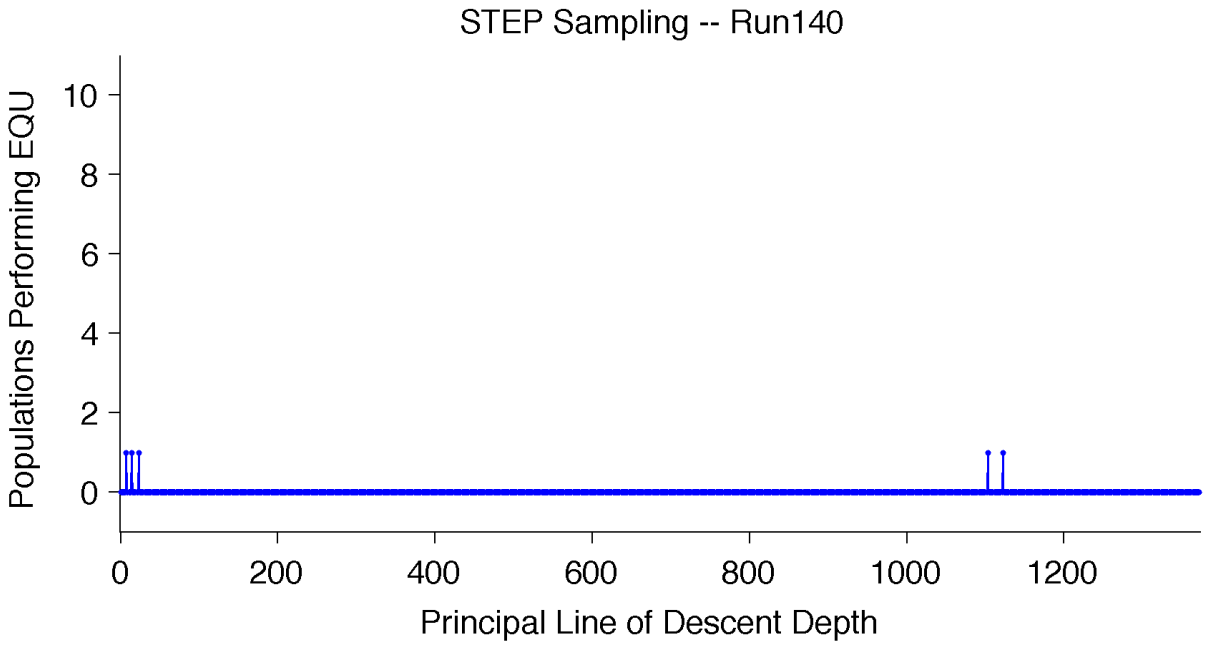

Figure B.40: STEP sampling along the PLoD from population 140.

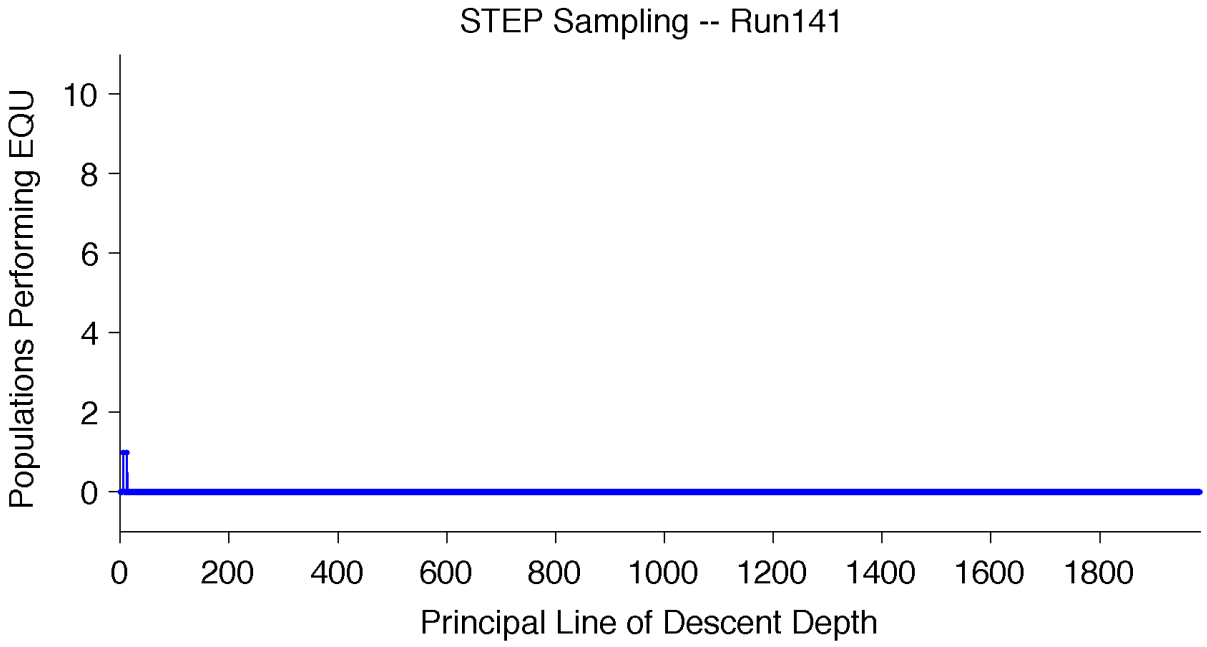

Figure B.41: STEP sampling along the PLoD from population 141.

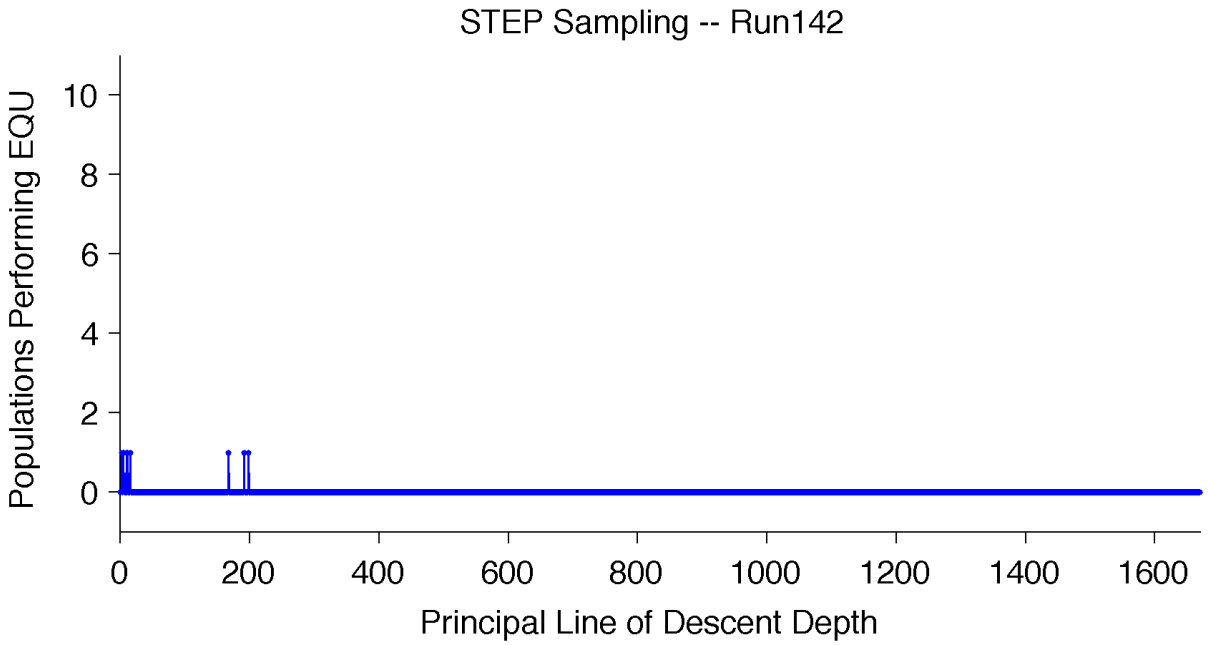

Figure B.42: STEP sampling along the PLoD from population 142.

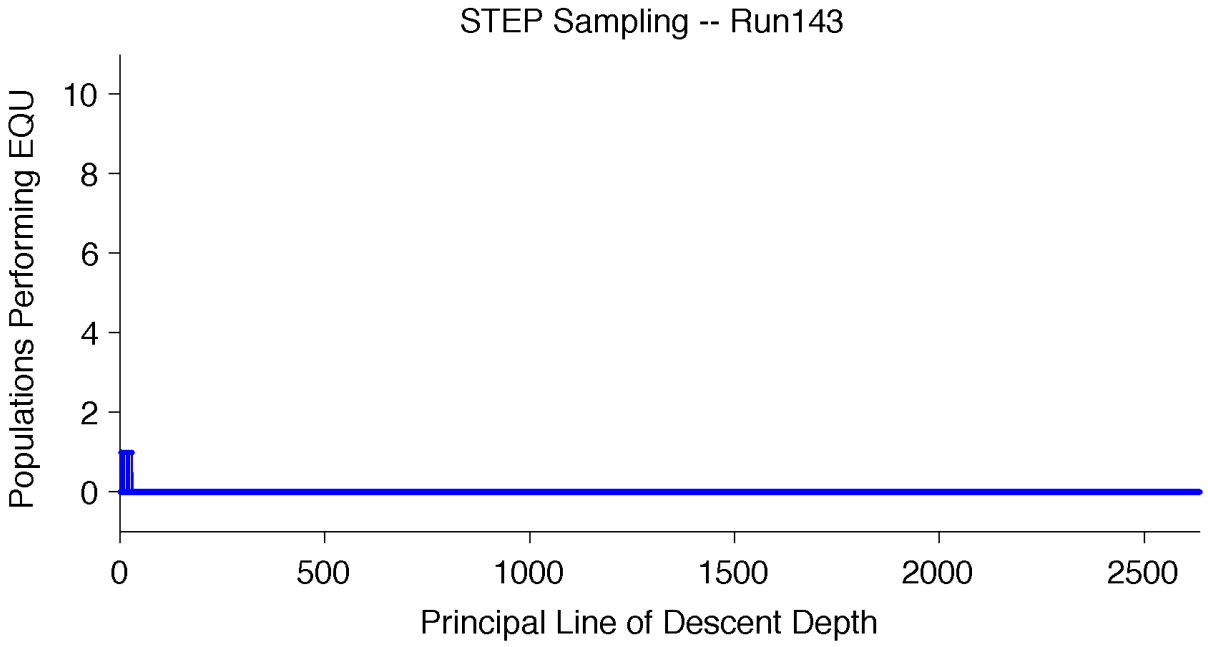

Figure B.43: STEP sampling along the PLoD from population 143.

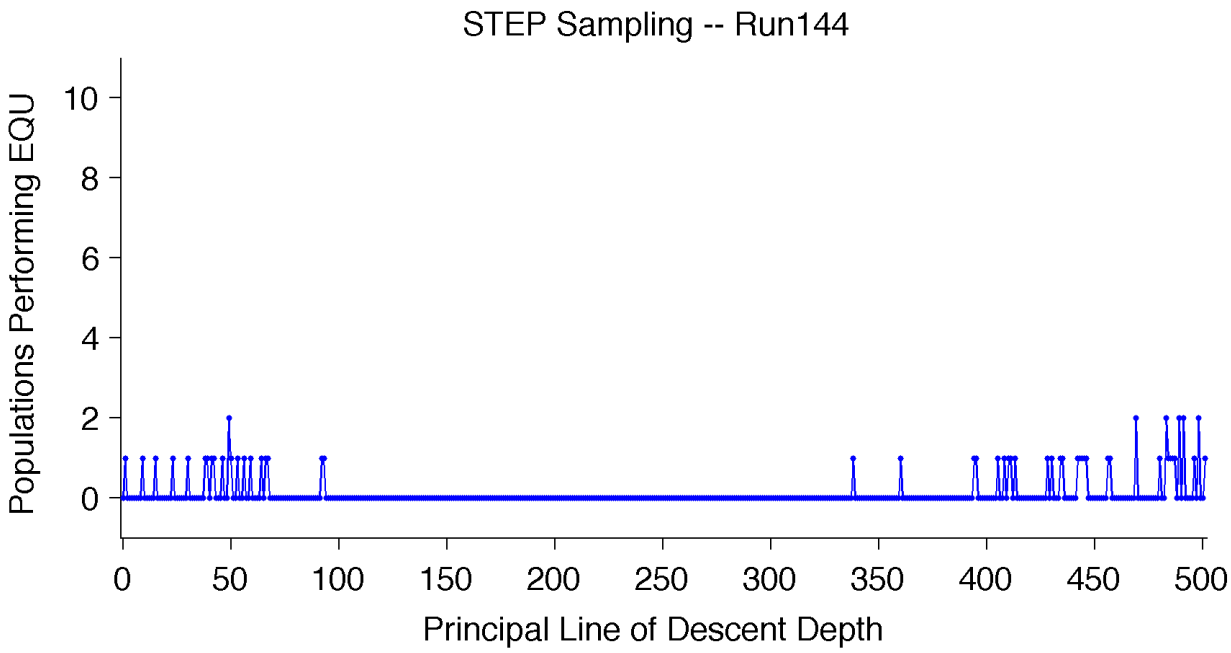

Figure B.44: STEP sampling along the PLoD from population 144.

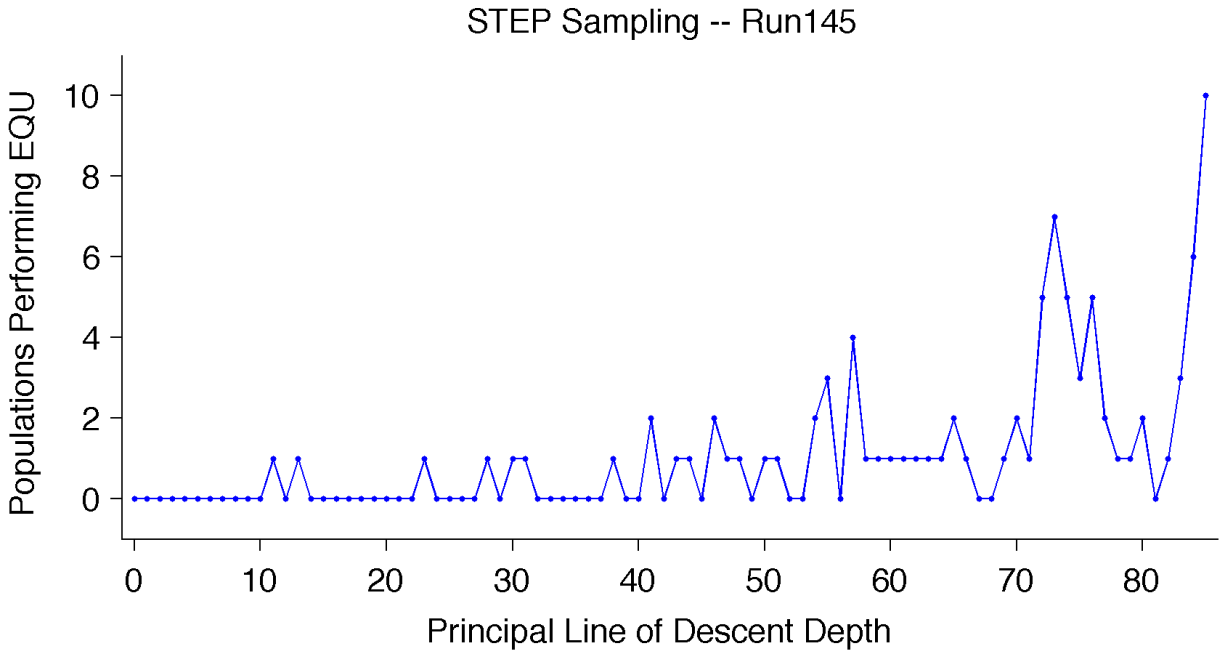

Figure B.45: STEP sampling along the PLoD from population 145.

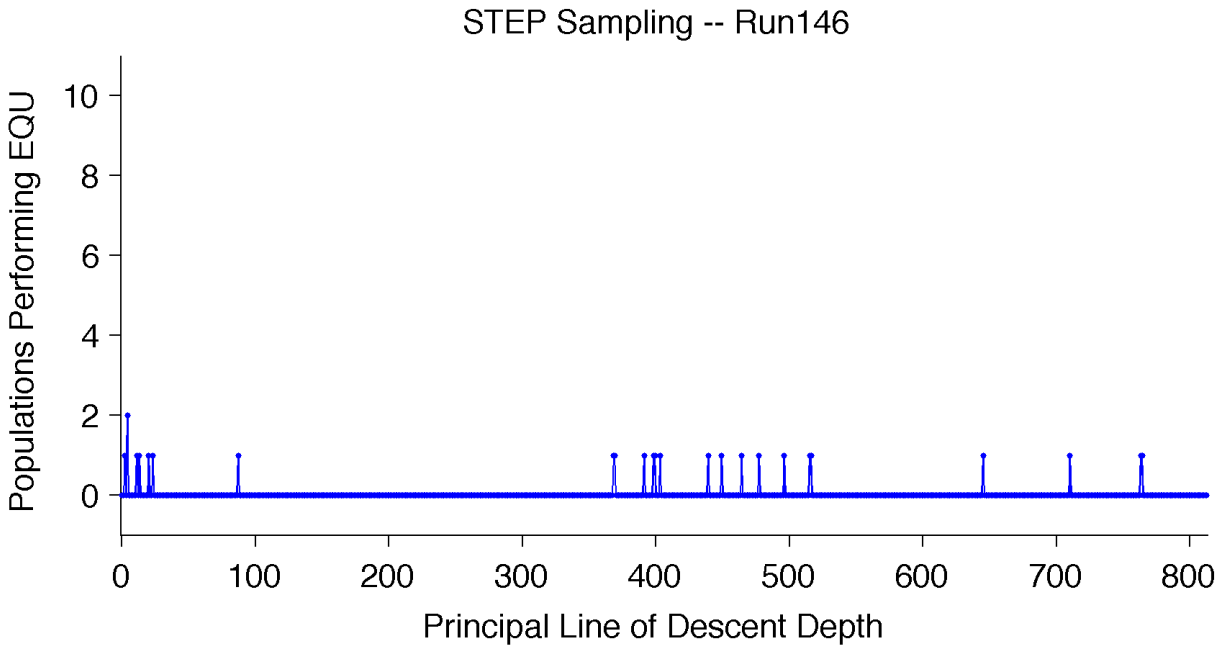

Figure B.46: STEP sampling along the PLoD from population 146.

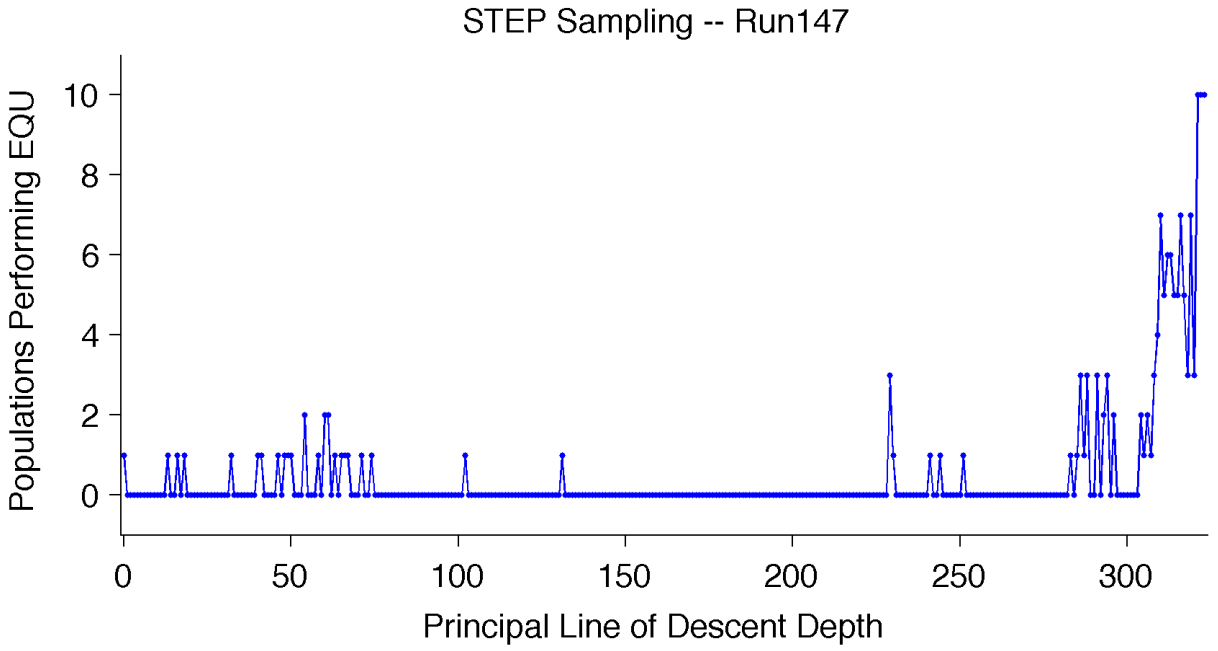

Figure B.47: STEP sampling along the PLoD from population 147.

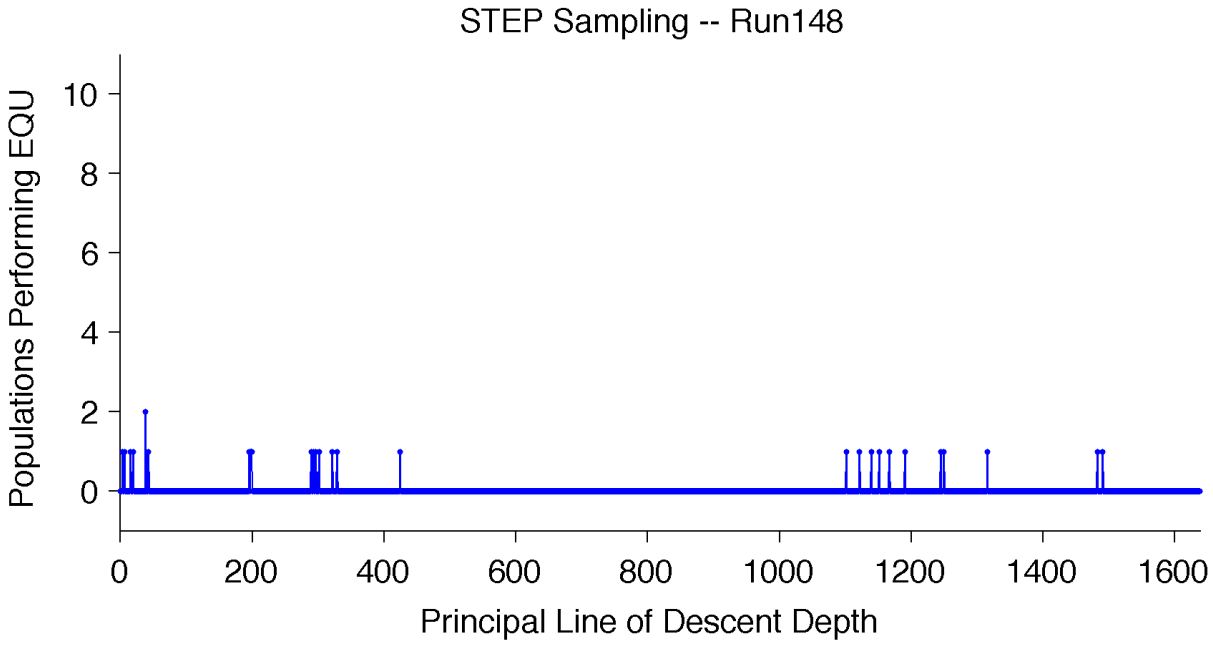

Figure B.48: STEP sampling along the PLoD from population 148.

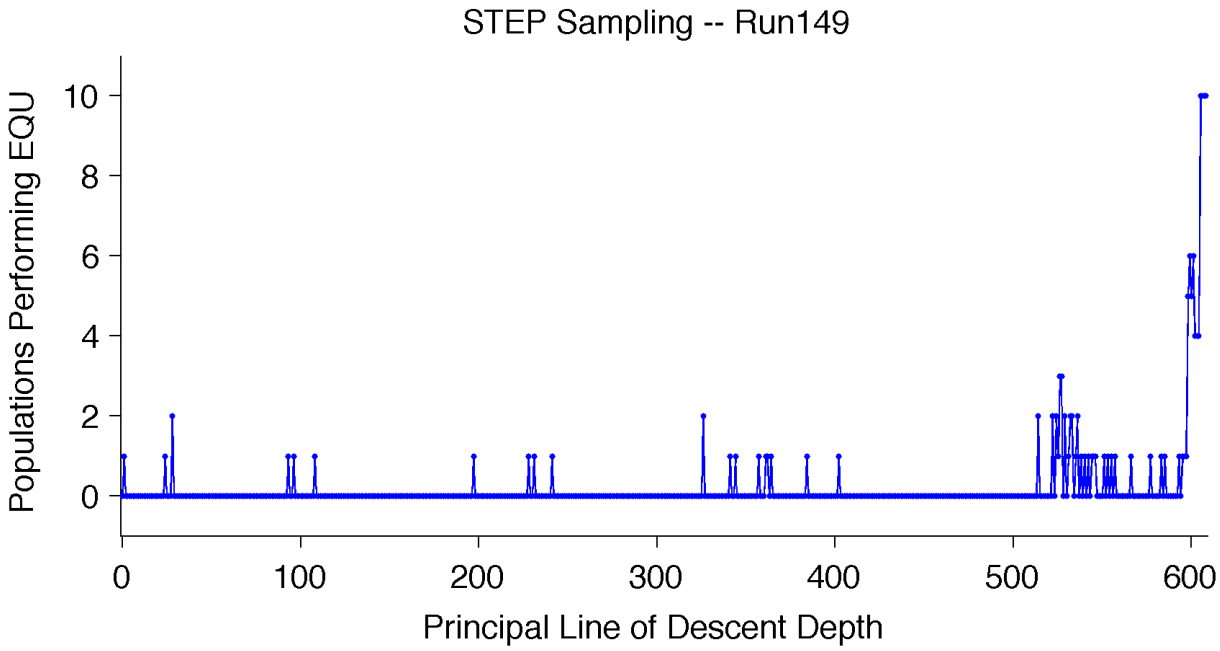

Figure B.49: STEP sampling along the PLoD from population 149.

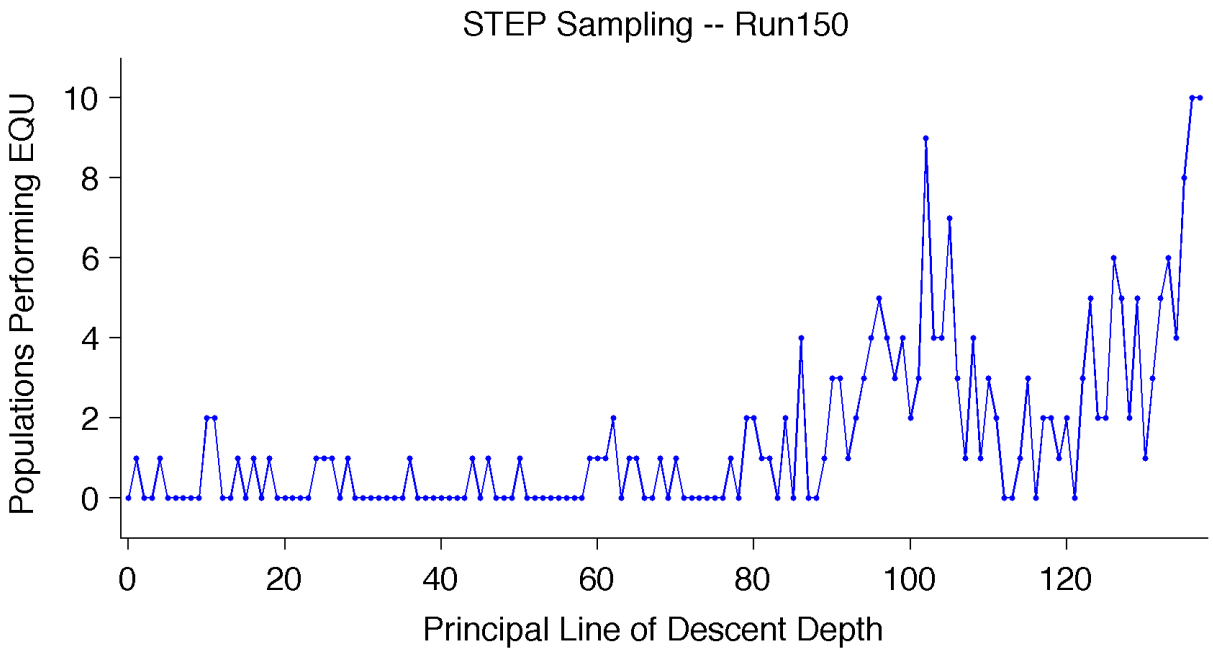

Figure B.50: STEP sampling along the PLoD from population 150.

References

References

Beatty, J. (2006). Replaying Life's tape. *Journal of Philosophy*, 103:336–362.

Blount, Z. D., Borland, C. Z., and Lenski, R. E. (2008). Historical contingency and the evolution of a key innovation in an experimental population of *Escherichia coli*. *Proceedings of the National Academy of Sciences*, 105:7899–7906.

Bryson, D. M. and Ofria, C. (2012). Digital Evolution Exhibits Surprising Robustness to Poor Design Decisions. In Adami, C., Bryson, D. M., Ofria, C., and Pennock, R. T., editors, *Artificial Life 13: Proceedings of the Thirteenth International Conference on the Simulation and Synthesis of Living Systems*, pages 19–26, East Lansing, MI.

Conner, J. K. and Hartl, D. L. (2004). *A Primer of Ecological Genetics*. Sinauer Associates, Sunderland, MA, 1st edition.

Conway Morris, S. (2003). *Life's Solution: Inevitable Humans in a Lonely Universe*. Cambridge University Press, Cambridge, UK.

Cooper, T. F. and Ofria, C. (2003). Evolution of stable ecosystems in populations of digital organisms. In Standish, R. K., Bedau, M. A., and Abbass, H. A., editors, *Artificial Life VIII: Proceedings of the Eighth International Conference on Artificial life*, pages 227–232, Cambridge, MA. International Society of Artificial Life, MIT Press.

Dawkins, R. (1996a). *Climbing Mount Improbable*. W. W. Norton & Company, New York.

Dawkins, R. (1996b). *The Blind Watchmaker*. W. W. Norton & Company, New York, third edition.

Eby, D., Averill, R. C., Punch, W. F., and Goodman, E. D. (1999). The Optimization of Flywheels using an Injection Island Genetic Algorithm. In Bentley, P., editor, *Evolutionary Design by Computers*, pages 167–190. Morgan Kaufmann, San Francisco, California, USA.

Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ.

Foote, M. (1998). Contingency and Convergence. *Science*, 280(5372):2068–2069.

Gould, S. J. (1989). *Wonderful Life: The Burgess Shale and the Nature of History*. W. W. Norton & Company, New York.

Grabowski, L. M., Bryson, D. M., Dyer, F. C., Pennock, R. T., and Ofria, C. (2011). Clever Creatures: Case Studies of Evolved Digital Organisms. In *Advances in Artificial Life, ECAL 2011: Proceedings of the Eleventh European Conference on the Synthesis and Simulation of Living Systems*, pages 276–283. MIT Press.

Grabowski, L. M., Bryson, D. M., Pennock, R. T., Dyer, F., and Ofria, C. (2010). Early evolution of memory usage in digital organism. In *Proceedings of the 12th International Conference on the Synthesis and Simulation of Living Systems*, pages 224–231, Odense, Denmark.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Koza, J. R. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, Norwell, MA.

Koza, J. R. and Poli, R. (2005). Genetic Programming. In Burke, E. and Kendal, G., editors, *Search Methodologies*. Springer.

Lenski, R., Ofria, C., Pennock, R. T., and Adami, C. (2003). The Evolutionary Origin of Complex Features. *Nature*, 423:139–144.

Lenski, R. E., Ofria, C., Collier, T. C., and Adami, C. (1999). Genome complexity, robustness and genetic interactions in digital organisms. *Nature*, 400(6745):661–664.

Losos, J. B., Jackman, T. R., Larson, A., de Queiroz, K., and Rodriguez-Schettino, L. (1998). Contingency and Determinism in Replicated Adaptive Radiations of Island Lizards. *Science*, 279:2115–2118.

Lynch, M. (2000). The Evolutionary Fate and Consequences of Duplicate Genes. *Science*, 290(5494):1151–1155.

Mayr, E. (1988). *Toward a New Philosophy of Biology: Observations of an Evolutionist*. Harvard University Press, Cambridge, MA, USA.

Misevic, D., Ofria, C., and Lenski, R. E. (2006). Sexual reproduction shapes the genetic architecture of digital organisms. *Proceedings of the Royal Society of London: Biological Sciences*, 273:457–464.

Ofria, C., Adami, C., and Collier, T. (2002). Design of Evolvable Computer Languages. *IEEE Transactions on Evolutionary Computation*, 6(4):420–424.

Ofria, C., Bryson, D. M., and Wilke, C. O. (2009). Avida: A Software Platform for Research in Computational Evolutionary Biology. In Adamatzky, A. and Komosinski, M., editors, *Artificial Life Models in Software*, pages 3–36. Springer-Verlag, London, UK.

Rechenberg, I. (1971). *Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution*. PhD thesis, Berlin Technical University.

Taylor, T. and Hallam, J. (1998). Replaying Life's Tape: An Investigation into the Role of Contingency in Evolution. In Adami, C., Belew, R., Kitano, H., and Taylor, C., editors, *Artificial Life VI: Proceedings of the Sixth International Conference on Artificial Life*, MA. International Society of Artificial Life, MIT Press.

Travisano, M., Mongold, J. A., Bennett, A. F., and Lenski, R. E. (1995). Experimental Tests of the Roles of Adaptation Chance and History in Evolution. *Science*, 267:87–90.

Van Valen, L. (1991). How far does contingency rule? *Evolutionary Theory*, 10:47–52.

Walker, B. and Ofria, C. (2012). Evolutionary Potential is Maximized at Intermediate Diversity Levels. In Adami, C., Bryson, D. M., Ofria, C., and Pennock, R. T., editors, *Artificial Life 13: Proceedings of the Thirteenth International Conference on the Simulation and Synthesis of Living Systems*, pages 116–120.

Weinreich, D. M. and Chao, L. (2005). Rapid Evolutionary Escape by Large Populations from Local Fitness Peaks is Likely in Nature. *Evolution*, 59(5):1175–1182.

Weinreich, D. M., Delaney, N. F., DePristo, M. A., and Hartl, D. L. (2006). Darwinian Evolution Can Follow Only Very Few Mutational Paths to Fitter Proteins. *Science*, 312(5770):111–114.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82.

Wolpert, D. H. and Macready, W. G. (1995). No Free Lunch Theorems for Search. Technical report.

Wright, S. (1988). Surfaces of Selective Value Revisited. *AMERICAN NATURALIST*, 131(1):115–123.

Yedid, G. and Bell, G. (2002). Macroevolution simulated with autonomously replicating computer programs. *Nature*, 420(6917):810–812.

Yedid, G., Ofria, C. A., and Lenski, R. E. (2009). Selective Press Extinctions, but Not Random Pulse Extinctions, Cause Delayed Ecological Recovery in Communities of Digital Organisms. *The American Naturalist*, 173(4):E139–E154.

Zhang, J. (2003). Evolution by gene duplication: an update. *Trends in Ecology and Evolution*, 18(6):292–298.