Digital Evolution Exhibits Surprising Robustness to Poor Design Decisions

David M. Bryson and Charles Ofria

Department of Computer Science and Engineering
BEACON Center for the Study of Evolution in Action
Michigan State University, East Lansing, MI 48824
brysonda@egr.msu.edu

Abstract

When designing an evolving software system, a researcher must set many aspects of the representation and inevitably make arbitrary decisions. Here we explore the consequences of poor design decisions in the development of a virtual instruction set in digital evolution systems. We evaluate the introduction of three different severities of poor choices. (1) functionally neutral instructions that water down mutational options, (2) actively deleterious instructions, and (3) a lethal `die` instruction. We further examine the impact of a high level of neutral bloat on the short term evolutionary potential of genotypes experiencing environmental change. We observed surprising robustness to these poor design decisions across all seven environments designed to analyze a wide range challenges. Analysis of the short term evolutionary potential of genotypes from the principal line of descent of case study populations demonstrated that the negative effects of neutral bloat in a static environment are compensated by retention of evolutionary potential during environmental change.

Introduction

Since the beginning of the field, evolutionary computation has taken its inspiration from biology. Genetic algorithms (Holland, 1975), genetic programming (Koza, 1990), and evolutionary strategies (Rechenberg, 1971) all exploit the power of mutation, selection, and differential survival to generate successful solutions to complex problems. A potential drawback of these traditional evolutionary computation techniques is that all methods require the researcher to define an explicit fitness function. All of the traits desired in the solution must be explicitly accounted for within the selection regime. As the complexity of the problem increases, this requirement becomes burdensome.

To address this and other challenges, researchers are taking further inspiration from biology and leveraging natural selection as instantiated by digital evolution (McKinley et al., 2008). Self-replicating computer programs, each running on their own virtual CPU, populate these digital evolution systems. Each program can be thought of as the genome of a digital organism, and consists of a string of instructions from a pre-defined set. To produce an offspring, a digital organism must copy its genome one line at a time, while being subject to environmental factors including other organisms and noise that causes errors (mutations) to this process. Since the digital organisms can interact and are responsible for their own replication, these systems have no explicit fitness function. The biggest power of the system is that it allows us to more easily translate concepts from natural biology. In order to direct evolution, an experimenter must craft an environment where the organisms face the same problem that the experimenter is trying to solve.

Researchers have used the Avida digital evolution system extensively to study evolutionary theory (Adami, 2006). Recent studies are pushing it into new, applied directions. For example, Knoester et al. (2007) and Beckmann et al. (2007) have explored communication and cooperation for distributed problem solving. Goldsby et al. (2007) investigate digital evolution as a tool for evolving software models for dynamic systems. Grabowski et al. (2008, 2010, 2011) study the evolution of movement and decision making. Many of these new experimental directions require changes to the virtual hardware and instruction set to support interaction with the environment and enhance the success of evolved solutions. The design of the instruction set architecture within an evolvable system can play an important role in the robustness and adaptability of evolved solutions (Ofria et al., 2002).

Changes to the instruction set may have a profound effect on the evolutionary potential of the system with respect to the environment. 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 we have investigated 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. We evaluate the evolutionary potential of each instruction set given a fixed amount of evolutionary time. In order to test the broad effect of each modification, we crafted seven computational environments representing a wide range of desired capabilities. We evalu-

ate the final results of experiments performed in each environment with each instruction set modification.

Given a fixed environment, a 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, organism genomes lock in features and genetic organization that are beneficial. The structure of these genomes impact their potential when the environment changes. We investigate this by evaluating short term evolutionary potential of genomes following environmental change. We examine how this potential changes relative to the progress of evolution in the origin environment.

Methods

We performed all experiments using Avida version 2.12.3[1]. We tested each instruction set architecture with 200 replicate populations in each of seven computational environments. We evolved the populations on a structured 60 x 60 toroidal grid for 100,000 updates[2]. Organisms were subject to a mutation rate of $2.5 * 10^{-3}$ per site in the genome, along with a $0.5 * 10^{-3}$ probability each for a single instruction insertion or deletion per site in the genome. Given that the ancestral organism had a length 100 genome, its mutation load was an average of 0.35 mutations per offspring, though size changes would change this load over time. All mutations, insertions, and deletions occurred upon division of the offspring. We seeded each population with a full complement of 3600 organisms with an ancestral genotype capable only of self-replication.

Instruction Sets

The HEADS architecture in Avida is the default virtual CPU configuration. The virtual hardware that implements this instruction set contains 26 commands designed to operate on a genomic program. It has three 32-bit registers, two stacks, four heads that point to positions in the genome, and input and output buffers. Among the 26 instructions in the set are three no-operation instructions, which can serve to modify the default behavior of other instructions, five flow-control instructions, three conditional instructions, seven arithmetic and logic instructions, five data movement instructions, and three instructions for self-replication.

The BLOAT instruction sets test what happens if you add too many useless, although not directly disruptive, instructions to the instruction set. They extend 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 three default no-operation instructions, it does not alter the behavior of other instructions. We tested four BLOAT instruction sets, varying the mutational frequency of the `nop-X` instruction. BLOAT-1 adds `nop-X` with a frequency of 1, yielding an effective mutational frequency of 0.037 for each instruction. BLOAT-3, BLOAT-10, and BLOAT-30 each increase the frequency of `nop-X` to 3, 10, and 30, respectively. In BLOAT-30, the effective mutational frequency of the nop-X instruction was 0.536, with 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 can actually disrupt 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% each time the organism executes the instruction. We hypothesized that lower penalties might be more detrimental to long term evolutionary, 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 it's lifetime, however, the organism will be immediately removed from the population.

Environment	Rewarded Functions
Logic-9	Nine 1- and 2-input logic operations.
Logic-77	Seventy-seven 1-, 2-, and 3-input logic operations.
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 1: The seven environments used to test instruction set modifications.

[1] Avida 2.12.3 source code is available for download, without cost, from http://avida.devosoft.org/. Specific instruction set configurations used are available upon request.

[2] An update is the natural unit of time in Avida, equal to an average of 30 instructions executed per living organism.

Environments

Avida supports a wide range of computational environments. We used seven distinct environments (Table 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.

The *Logic-9* environment consists of metabolic rewards for all possible 1- and 2-input binary logic operations; there are 9 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 organisms' 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)

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.

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'.

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.

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 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.

The *Limited-9* environment, based on the Logic-9 environment, offers metabolic rewards for all possible 1- and 2-input binary 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).

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.

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 configured similarly to the settings used in the experiment that was the source of the genotype. We then allow the world to evolve for a short period, $10,000$ updates (approximately $1,000$ generations) for the work presented here, after which we collect metrics of interest, such as phenotype and fitness. We repeat this procedure with the same reference genotype multiple times for statistical assessment of the genotype's evolutionary potential.

General Performance Evaluation

We have focused on two measures of evolved populations to evaluate the general performance of 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 fin-

	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 2: 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 ($p < 0.05$) 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)**	$\mathbf{1.01 \times 10^{-4}}$ **(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)**	$\mathbf{9.7 \times 10^{-5}}$ **(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 3: 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 ($p < 0.05$) deviations after sequential Bonferroni correction.

ished. 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 the control (HEADS) 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,

$$t_p = \sum_{i=1}^{N_p} \sum_{j=1}^{T} \frac{q_{i,j}}{N_p} \quad (1)$$

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.

Upon analysis, all three POISON instruction sets and the DIE instruction set demonstrated no significant variation in either population mean fitness (Table 2) or task success (Table 3) 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 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.

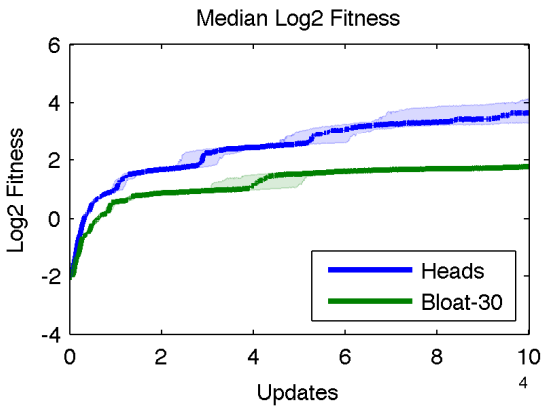

Figure 1: Median $\log_2$ fitness trajectory of the HEADS (blue line) and BLOAT-30 (green line) architectures. Lines calculated from all 200 replicates of each instruction set in the Fibonacci-32 environment. Shaded regions show 95% bootstrap confidence intervals, 10,000 iterations.

Impact on Evolutionary Potential

The Bloat 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 1 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 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 will 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, the complete lineage of ancestral genotypes that gave rise to the final, numerically dominant genotype, 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 2). 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).

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 the Logic-9 source environment within the Fibonacci-32 sample environment (see Figure 3). 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). 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).

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 dig-

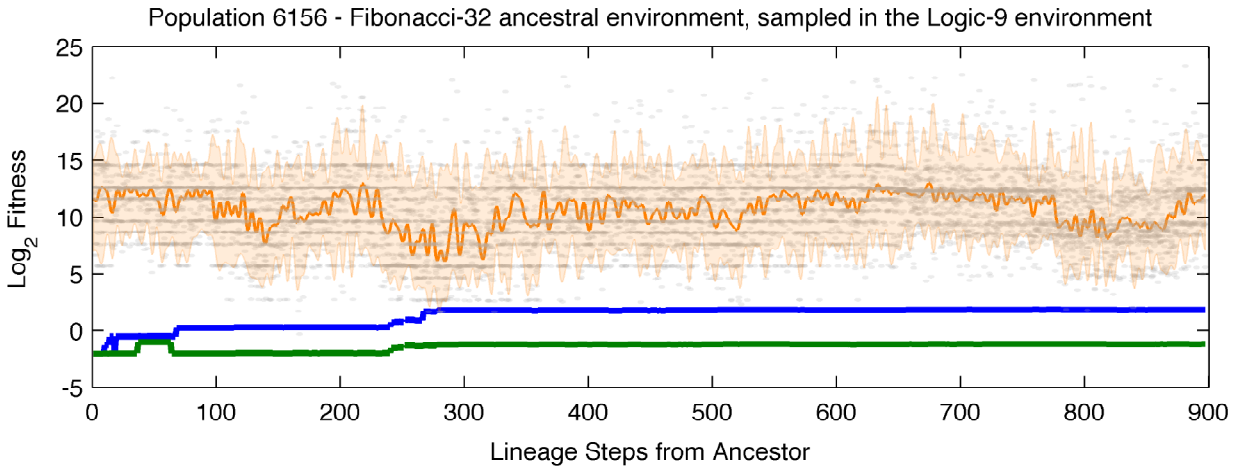

Figure 2: STEP sampling results of population 6156 originally evolved in the Fibonacci-32 environment with the HEADS instruction set. Blue line: fitness of the reference genotype in the Fibonacci-32 environment. Green line: initial fitness of the reference genotype in the Logic-9 environment. Orange 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.

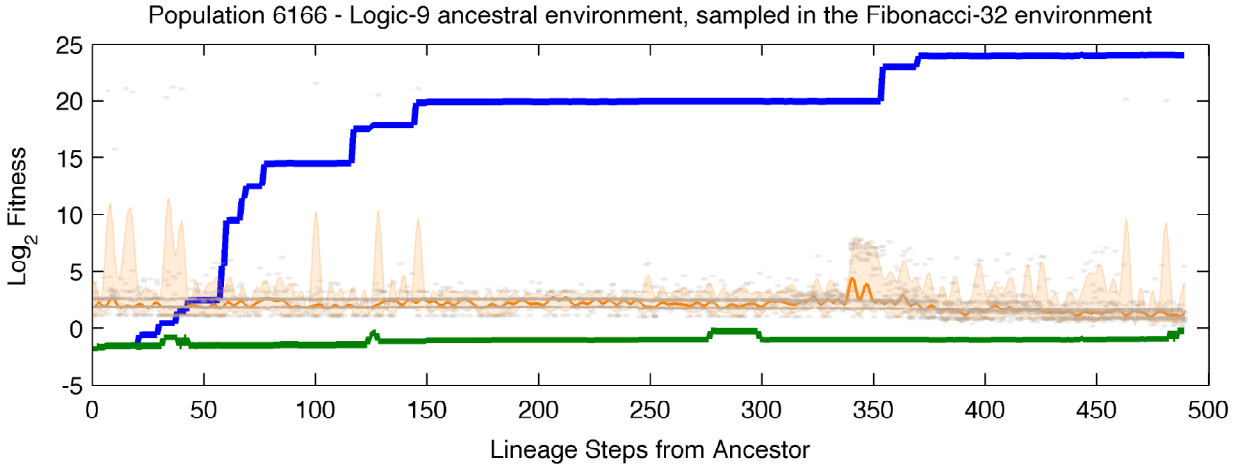

Figure 3: STEP sampling results of population 6166 originally evolved in the Logic-9 environment with the HEADS instruction set. Blue line: fitness in the Logic-9 environment. Green line: initial fitness in the Fibonacci-32 environment. Orange line: median STEP results (smoothed) with 95% quantiles. Gray circles: individual STEP results.

ital 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 on 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 the 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 evolved with the

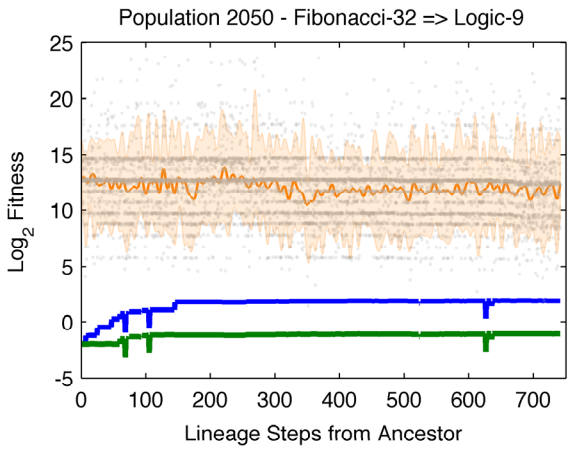

Figure 4: Fitness STEP sampling results of population 6166 originally evolved in the Logic-9 environment with the BLOAT-30 instruction set. Blue line: fitness in the Logic-9 environment. Green line: initial fitness in the Fibonacci-32 environment. Orange line: median STEP results (smoothed) with 95% quantiles. Gray circles: individual STEP results.

Bloat-30 instruction set 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.

Acknowledgements

The authors wish to thank Heather Goldsby for helpful discussion and Richard Lenski for guidance in developing STEP sampling. 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.

References

Adami, C. (2006). Digital genetics: unravelling the genetic basis of evolution. *Nature Reviews Genetics*, 7(2):109–118.

Beckmann, B., McKinley, P. K., Knoester, D. B., and Ofria, C. (2007). Evolution of Cooperative Information Gathering in Self-Replicating Digital Organisms. In *Proceedings of the First IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, Boston, Massachusetts.

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.

Goldsby, H. J., Knoester, D. B., Cheng, B. H. C., McKinley, P. K., and Ofria, C. A. (2007). Digitally Evolving Models for Dynamically Adaptive Systems. In *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Minneapolis, Minnesota.

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 Internation Conference on the Synthesis and Simulation of Living Systems*, pages 224–231.

Grabowski, L. M., Elsberry, W. R., Ofria, C. A., and Pennock, R. T. (2008). On the Evolution of Motility and Intelligent Tactic Response. In *GECCO '08: Proceedings of the 2008 conference on Genetic and evolutionary computation*.

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

Knoester, D. B., McKinley, P. K., Beckmann, B., and Ofria, C. A. (2007). Directed Evolution of Communication and Cooperation in Digital Organisms. In *Proceedings of the 9th European Conference on Artificial Life*, Lisbon, Portugal. Springer.

Koza, J. R. (1990). Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems. Technical Report STAN-CS-90-1314.

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.

McKinley, P. K., Cheng, B. H. C., Ofria, C., Knoester, D., Beckmann, B., and Goldsby, H. (2008). Harnessing Digital Evolution. *IEEE Computer*, 41(1).

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.

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

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.