Evolving a New Feature for a Working Program

Mike Stimpson

January 18, 2013

Abstract
A genetic programming system is created. A first fitness function \( f_1 \) is used to evolve a program that implements a first feature. Then the fitness function is switched to a second function \( f_2 \), which is used to evolve a program that implements a second feature while still maintaining the first feature. The median number of generations \( G_1 \) and \( G_2 \) needed to evolve programs that work as defined by \( f_1 \) and \( f_2 \) are measured. The behavior of \( G_1 \) and \( G_2 \) are observed as the difficulty of the problem is increased. In these systems, the density \( D_1 \) of programs that work (for fitness function \( f_1 \)) is measured in the general population of programs. The relationship \( G_1 \sim \frac{1}{\sqrt{D_1}} \) is observed to approximately hold. Also, the density \( D_2 \) of programs that work (for fitness function \( f_2 \)) is measured in the general population of programs. The relationship \( G_2 \sim \frac{1}{\sqrt{D_2}} \) is observed to approximately hold.

1 INTRODUCTION

Previous work [1] demonstrated that, when evolving a program from random starting programs, the relationship \( G \sim \frac{1}{\sqrt{D}} \) often approximately held, where \( G \) was the median number of generations required to evolve a working program, and \( D \) was the density of working programs in the general population of programs. This paper examines what happens when, after evolving a first feature, we attempt to evolve a second feature of equivalent complexity.

The rest of this paper is organized as follows: Section 2 describes the system. Section 3 presents the results in the form of several data sets. Section 4 demonstrates the relationship between the density of working programs and the median number of generations needed to evolve a working program. Section 5 presents some conclusions, and section 6 presents some open questions.

2 THE SYSTEM: TREE-STRUCTURED PROGRAMS, SORTING INTEGERS

I chose sorting a list of integers as the problem that programs were attempting to solve. The first feature was sorting in ascending order; the second feature was sorting in descending order.
The system had a fixed number $v$ of writable variables, numbered 1 through $v$. (*Fixed* here means that it did not evolve; however, it could be changed between runs via a command-line parameter.) It also contained three read-only "variables". Variable 0 always contained 0. Variable $v+1$ always contained the number of integers in the list being sorted. Variable $v+2$ contained 0 if the list was to be sorted in ascending order, and 1 otherwise.

A program was represented as LISP-like tree structure. The trees were limited to a maximum depth of 6. Programs contained a variable number of nodes. Mutations could alter a whole sub-tree, rather than a single node.

However, the programs were not purely in the LISP style, in that each node could access any of the variables - variables were not sub-nodes of operator nodes.

Statements were created from the following node types: For, IfElse, CompareSwap, and ReverseCompareSwap. CompareSwap and ReverseCompareSwap were leaf nodes; For and IfElse were not.

For was a C-style for loop with a loop variable, a variable from which to initialize the loop variable, and a limit variable to compare the loop variable to. It required one child node, which it executed once for each iteration of the loop.

IfElse was an if/else on a variable. It required two child nodes. If the variable was non-zero, it executed the "if" node; otherwise, it executed the "else" node. Either the "if" node or the "else" node (or both) could be null (no operation).

CompareSwap compared two numbers in the list, and swapped them if they were out of ascending order. ReverseCompareSwap was identical, except that it swapped them if they were out of descending order.

The difficulty was changed by increasing the number of variables, which increased the odds of using the wrong variables when attempting to create nested loops. That is, it decreased the probability of creating a working program.

The population size was 1000 programs. Parents were chosen by a 7-way tournament of randomly-chosen programs.

The fitness function $f_1$ was computed by having each program attempt to sort three lists of numbers, which contained 10, 30 and 50 values. The lists contained the values from 1 to the size of the list, in random order. After a program attempted to sort a list, the *forward distance* was computed as follows: For each location in the list, the absolute value was taken of the difference between the value at that location in the list as sorted by the program, and the value that would be at that location if the list were perfectly sorted in ascending order. A perfectly sorted list therefore had a forward distance of zero. The *reverse distance* was identical, except that the perfectly sorted list was replaced by one that was perfectly sorted in reverse (descending) order. In general, the forward and backward distances were larger for the longer lists. To address this, a *normalized metric* was created for each list, which was the reverse distance minus the forward distance, divided by the sum of the forward and reverse distances. This evaluated to 1 for a list perfectly sorted in ascending order, and to -1 for a list that was perfectly sorted in descending order. Finally, $f_1$ was the average of the normalized metrics for the three lists.
The fitness function $f_2$ ran the program twice, once with variable $v + 2$ cleared, and once with it set. For both runs, $f_1$ was computed on the results. Call the results $f_{1a}$ and $f_{1d}$ for the runs that should sort in ascending and descending order, respectively. Then, $f_{1d} = -1$ means that the program sorted perfectly in descending order, and $f_{1d} = 1$ means that the program failed as badly as possible to sort in descending order. Then $f_2 = \frac{f_{1a} - f_{1d}}{2}$ yields 1 if the lists were sorted correctly in both ascending and descending order, and -1 if they were always sorted in the wrong order.

I also used the fitness function $f_3 = \frac{(2f_{1a} - f_{1d})}{3}$. This was similar to $f_2$, but it placed greater weight on preserving the ability to sort in ascending order (that is, to preserve the functionality that was already evolved by using $f_1$).

If the program executed 10 times as many statements as bubble sort would require for the same list, the program was considered to be in a semi-infinite loop, and terminated. No fitness penalty was imposed for this condition.

The unsorted lists of numbers were randomly created. New lists were created for each generation. The same lists were used for all programs of any one generation.

After evolving a working program according to metric $f_1$, the evolution was continued for ten more generations using $f_1$, in order to reach something approaching a steady state, and yet not to reach a monoculture. After these ten generations, approximately 96% of the programs had a perfect fitness function according to $f_1$. Then the fitness function was switched to $f_2$ or $f_3$.

An evolution started with a random collection of programs, and proceeded until a program evolved that worked (had a fitness function of 1.0). An evolution was characterized by the number of generations required to evolve a working program as determined by $f_1$, and the number of generations needed to evolve a working program according to $f_2$ or $f_3$. The ten generations to approach steady state were not included in these numbers. Also, the number of generations for $f_2$ or $f_3$ did not include the generations when the fitness function was $f_1$.

However, since evolution is a random process, a repeat of the evolution would take a completely different number of generations.

A run was 100 evolutions, all with the same parameters. It was characterized by the median of the number of generations required for the evolutions in the run. $G_1$ was the median number of generations with fitness function $f_1$ (excluding the ten generations to approach steady state); $G_2$ was the median number of generations with fitness function $f_2$ or $f_3$. (The distribution of the number of generations had a very long tail. The presence or absence of one anomalous evolution could significantly shift the average, so the median was the appropriate choice here.)

I also measured the density of working programs (as defined by $f_1$ or $f_2$ - note that $f_3$ gives the same definition of "working" as $f_2$) in the general population of programs, by generating a large number of random programs and seeing how many of them worked as is, that is, with no evolution. I made sure that the sample was large enough to contain at least 100 working programs.

The system presented a problem when measuring densities, because the universe of all possible programs is not, in general, very much like the set of programs that work. The universe of all
possible programs is weighted heavily toward the longest lengths, but the working programs are not. As an evolution proceeds, the length distribution of the population of programs should become more and more similar to the distribution of working programs, and less and less similar to the distribution of the universe of all possible programs. Given, then, that the universe of all possible programs is structurally different from both the working programs that are evolved and from the population during an evolution, how can we get meaningful density data? I chose the approach of trying to create self-consistent population distributions - that is, population distributions such that, when populations with that length distribution were evolved, the resulting working programs had the same distribution of lengths. (In practice, this could only be approximately achieved.) If we measure the density of a population of programs with the same length distribution as the working programs, we obtain density data that we can meaningfully combine with the median number of generations. (The alternative - the density data coming from populations that are unlike the population of working programs - clearly is less likely to provide meaningful data.)

Finally, I measured the density of programs that worked as defined by $f_2$ within the set of programs that worked as defined by $f_1$. Again, I made sure that the sample was large enough to contain at least 100 working programs (as defined by $f_2$).

3 DATA AND ANALYSIS

Generations to evolve a working program, using metric $f_2$:

| Number of variables | $G_1$ | $G_2$ |
|---------------------|-------|-------|
| 2                   | 1     | 62.5  |
| 3                   | 4     | 82.5  |
| 4                   | 5     | 175   |
| 5                   | 7     | 129.5 |
| 6                   | 16    | 212.5 |
| 7                   | 21    | 239   |
| 8                   | 38.5  | 458   |
| 9                   | 51    | 720   |
| 10                  | 78    | 462.5 |

A second try with the same parameters:

| Number of variables | $G_1$ | $G_2$ |
|---------------------|-------|-------|
| 2                   | 1     | 72.5  |
| 3                   | 3     | 92.5  |
| 4                   | 5     | 84.5  |
| 5                   | 7     | 228.5 |
| 6                   | 11    | 272.5 |
| 7                   | 30    | 349.5 |
| 8                   | 35    | 346.5 |
| 9                   | 53.5  | 337.5 |
| 10                  | 57.5  | 463   |
Generations to evolve a working program, using metric \( f_3 \):

| Number of variables | \( G_1 \) | \( G_2 \) |
|---------------------|-----------|-----------|
| 2                   | 1         | 60.5      |
| 3                   | 2.5       | 107.5     |
| 4                   | 5         | 255       |
| 5                   | 6         | 203.5     |
| 6                   | 16.5      | 410.5     |
| 7                   | 21        | 470       |
| 8                   | 43        | 613.5     |
| 9                   | 67        | 718.5     |
| 10                  | 104.5     | 863       |

A second try with the same parameters:

| Number of variables | \( G_1 \) | \( G_2 \) |
|---------------------|-----------|-----------|
| 2                   | 1         | 69.5      |
| 3                   | 1         | 105       |
| 4                   | 4         | 152.5     |
| 5                   | 6         | 292       |
| 6                   | 11        | 196       |
| 7                   | 27.5      | 457.5     |
| 8                   | 28        | 379.5     |
| 9                   | 46        | 859       |
| 10                  | 74.5      | 794.5     |

The different \( G_1 \) values between the data for \( f_2 \) and \( f_3 \) are statistical fluctuations. In all cases, \( G_1 \) was for programs that were evolved using metric \( f_1 \). (Clearly the data contains a lot of noise!)

\( f_3 \) took more generations than \( f_2 \) to evolve the same program. This seems intuitively reasonable, since \( f_3 \) places a higher value on preserving the existing functionality.

What happens if we don’t use metric \( f_1 \) to evolve a solution to a sub-problem? What if we just use metric \( f_2 \) or \( f_3 \) the whole way? Let us call the median number of generations \( G'_{2} \).

Generations to evolve a working program, using metric \( f_2 \) only:

| Number of variables | \( G'_{2} \) |
|---------------------|-------------|
| 2                   | 72.5        |
| 3                   | 67          |
| 4                   | 131.5       |
| 5                   | 247         |
| 6                   | 397         |
| 7                   | 462.5       |
| 8                   | 651.5       |
| 9                   | 1003        |
| 10                  | 1318.5      |
Generations to evolve a working program, using metric \( f_3 \) only:

| Number of variables | \( G' \) |
|---------------------|--------|
| 2                   | 35.5   |
| 3                   | 108    |
| 4                   | 138    |
| 5                   | 206    |
| 6                   | 258    |
| 7                   | 288.5  |
| 8                   | 469    |
| 9                   | 650.5  |
| 10                  | 649    |

Clearly, trying to evolve a working program using only \( f_2 \) took more total generations than using \( f_1 \) and then \( f_2 \), but using \( f_3 \) only took fewer total generations than using \( f_1 \) and then \( f_3 \). A possible reason for this is that \( f_2 \) is symmetric - an initial random program is not likely to sort (even partially) in both ascending and descending order, and a program that (partially or completely) sorts only in ascending (or descending) order gets a fitness of zero according to \( f_2 \). But \( f_3 \) gives a positive value for a program that sorts (even partially) in ascending order only. Programs can therefore begin evolving under \( f_3 \) more easily than under \( f_2 \).

Density \( D_1 \) of fully-working programs (as measured by \( f_1 \)) in the general population of programs:

| Number of variables | \( D_1 \) |
|---------------------|--------|
| 2                   | \( 1.107 \times 10^{-3} \) |
| 3                   | \( 5.9 \times 10^{-4} \) |
| 4                   | \( 3.1 \times 10^{-4} \) |
| 5                   | \( 1.61 \times 10^{-4} \) |
| 6                   | \( 1.16 \times 10^{-4} \) |
| 7                   | \( 6.0 \times 10^{-5} \) |
| 8                   | \( 4.3 \times 10^{-5} \) |
| 9                   | \( 3.45 \times 10^{-5} \) |
| 10                  | \( 2.06 \times 10^{-5} \) |
Density $D_2$ of fully-working programs (as measured by $f_2$ or $f_3$) in the general population of programs:

| Number of variables | $D_2$       |
|---------------------|-------------|
| 2                   | $4.4 \times 10^{-6}$ |
| 3                   | $1.07 \times 10^{-6}$ |
| 4                   | $3.3 \times 10^{-7}$ |
| 5                   | $1.06 \times 10^{-7}$ |
| 6                   | $5.37 \times 10^{-8}$ |
| 7                   | $1.96 \times 10^{-8}$ |
| 8                   | $1.06 \times 10^{-8}$ |
| 9                   | $5.87 \times 10^{-9}$ |
| 10                  | $2.47 \times 10^{-9}$ |

But the evolution using metric $f_2$ was not done on a collection of random programs; it was done on programs almost all of which were fully working as defined by metric $f_1$. Perhaps, then, rather than using $D_2$ (the density of programs that are fully working under metric $f_2$ within the general population of programs), we should use the density of programs that are fully working under metric $f_2$ within the population of programs that are fully working under metric $f_1$. Call this density $D'_2$.

| Number of variables | $D'_2$     |
|---------------------|------------|
| 2                   | $4.31 \times 10^{-6}$ |
| 3                   | $2.03 \times 10^{-3}$ |
| 4                   | $1.111 \times 10^{-3}$ |
| 5                   | $6.95 \times 10^{-4}$ |
| 6                   | $4.68 \times 10^{-4}$ |
| 7                   | $2.86 \times 10^{-4}$ |
| 8                   | $2.21 \times 10^{-4}$ |
| 9                   | $1.594 \times 10^{-4}$ |
| 10                  | $1.244 \times 10^{-4}$ |

4 RELATIONSHIP BETWEEN SOLUTION DENSITY AND NUMBER OF GENERATIONS

Combining the measured densities with the median number of generations to reach a working program, we observe a pattern: As we change the number of variables, the median number of generations needed to evolve a working program is almost proportional to the reciprocal of the square root of the density; that is, $K_1 = G_1 \times \sqrt{D_1}$ is almost constant. This value ($K_1$) rises slowly as $D_1$ decreases. But $K_2 = G_2 \times \sqrt{D_2}$ decreases slowly as $D_2$ decreases.
Evolved using metric $f_2$:

| Number of variables | $G_1$ | $D_1$          | $K_1$ | $G_2$ | $D_2$         | $K_2$ |
|---------------------|-------|----------------|-------|-------|---------------|-------|
| 2                   | 1     | $1.107 \times 10^{-3}$ | 0.0333 | 62.5  | $4.4 \times 10^{-6}$ | 0.1311 |
| 3                   | 4     | $5.9 \times 10^{-4}$   | 0.0972 | 82.5  | $1.07 \times 10^{-6}$ | 0.0853 |
| 4                   | 5     | $3.1 \times 10^{-4}$   | 0.088  | 175   | $3.3 \times 10^{-7}$  | 0.1005 |
| 5                   | 7     | $1.61 \times 10^{-4}$  | 0.0888 | 129.5 | $1.06 \times 10^{-7}$ | 0.0422 |
| 6                   | 16    | $1.16 \times 10^{-4}$  | 0.1723 | 212.5 | $5.37 \times 10^{-8}$ | 0.0492 |
| 7                   | 21    | $6.0 \times 10^{-5}$   | 0.1627 | 239   | $1.962 \times 10^{-8}$ | 0.0353 |
| 8                   | 38.5  | $4.3 \times 10^{-5}$   | 0.252  | 458   | $1.06 \times 10^{-8}$ | 0.0472 |
| 9                   | 51    | $3.45 \times 10^{-5}$  | 0.3    | 720   | $5.87 \times 10^{-9}$ | 0.0551 |
| 10                  | 78    | $2.06 \times 10^{-5}$  | 0.354  | 462.5 | $2.47 \times 10^{-9}$ | 0.023  |

A second try with the same parameters:

| Number of variables | $G_1$ | $D_1$          | $K_1$ | $G_2$ | $D_2$         | $K_2$ |
|---------------------|-------|----------------|-------|-------|---------------|-------|
| 2                   | 1     | $1.107 \times 10^{-3}$ | 0.0333 | 72.5  | $4.4 \times 10^{-6}$ | 0.1521 |
| 3                   | 3     | $5.9 \times 10^{-4}$   | 0.0729 | 92.5  | $1.07 \times 10^{-6}$ | 0.0957 |
| 4                   | 5     | $3.1 \times 10^{-4}$   | 0.088  | 84.5  | $3.3 \times 10^{-7}$  | 0.0485 |
| 5                   | 7     | $1.61 \times 10^{-4}$  | 0.0888 | 228.5 | $1.06 \times 10^{-7}$ | 0.0744 |
| 6                   | 11    | $1.16 \times 10^{-4}$  | 0.1184 | 272.5 | $5.37 \times 10^{-8}$ | 0.0631 |
| 7                   | 30    | $6.0 \times 10^{-5}$   | 0.232  | 349.5 | $1.962 \times 10^{-8}$ | 0.049  |
| 8                   | 35    | $4.3 \times 10^{-5}$   | 0.23   | 346.5 | $1.06 \times 10^{-8}$ | 0.0357 |
| 9                   | 53.5  | $3.45 \times 10^{-5}$  | 0.314  | 337.5 | $5.87 \times 10^{-9}$ | 0.0259 |
| 10                  | 57.5  | $2.06 \times 10^{-5}$  | 0.261  | 463   | $2.47 \times 10^{-9}$ | 0.023  |

Evolved using metric $f_3$:

| Number of variables | $G_1$ | $D_1$          | $K_1$ | $G_2$ | $D_2$         | $K_2$ |
|---------------------|-------|----------------|-------|-------|---------------|-------|
| 2                   | 1     | $1.107 \times 10^{-3}$ | 0.0333 | 60.5  | $4.4 \times 10^{-6}$ | 0.1269 |
| 3                   | 2.5   | $5.9 \times 10^{-4}$   | 0.0607 | 107.5 | $1.07 \times 10^{-6}$ | 0.1112 |
| 4                   | 5     | $3.1 \times 10^{-4}$   | 0.088  | 255   | $3.3 \times 10^{-7}$  | 1465  |
| 5                   | 6     | $1.61 \times 10^{-4}$  | 0.0761 | 203.5 | $1.06 \times 10^{-7}$ | 0.0663 |
| 6                   | 16.5  | $1.16 \times 10^{-4}$  | 0.1777 | 410.5 | $5.37 \times 10^{-8}$ | 0.0951 |
| 7                   | 21    | $6.0 \times 10^{-5}$   | 0.1627 | 470   | $1.962 \times 10^{-8}$ | 0.0658 |
| 8                   | 43    | $4.3 \times 10^{-5}$   | 0.282  | 613.5 | $1.06 \times 10^{-8}$ | 0.0632 |
| 9                   | 67    | $3.45 \times 10^{-5}$  | 0.394  | 718.5 | $5.87 \times 10^{-9}$ | 0.055  |
| 10                  | 104.5 | $2.062 \times 10^{-5}$ | 0.475  | 863   | $2.47 \times 10^{-9}$ | 0.0429 |
A second try with the same parameters:

| Number of variables | $G_1$  | $D_1$  | $K_1$  | $G_2$  | $D_2$  | $K_2$  |
|---------------------|--------|--------|--------|--------|--------|--------|
| 2                   | 1      | $1.107 \times 10^{-3}$ | 0.0333 | 69.5   | $4.4 \times 10^{-6}$ | 0.1458 |
| 3                   | 1      | $5.9 \times 10^{-4}$ | 0.0243 | 105    | $1.07 \times 10^{-6}$ | 0.1086 |
| 4                   | 4      | $3.1 \times 10^{-4}$ | 0.0704 | 152.5  | $3.3 \times 10^{-7}$ | 0.0876 |
| 5                   | 6      | $1.61 \times 10^{-4}$ | 0.0761 | 292    | $1.06 \times 10^{-7}$ | 0.0951 |
| 6                   | 11     | $1.16 \times 10^{-4}$ | 0.1185 | 196    | $5.37 \times 10^{-8}$ | 0.0454 |
| 7                   | 27.5   | $6.0 \times 10^{-5}$ | 0.213  | 457.5  | $1.962 \times 10^{-8}$ | 0.0641 |
| 8                   | 28     | $4.3 \times 10^{-5}$ | 0.1836 | 379.5  | $1.06 \times 10^{-8}$ | 0.0391 |
| 9                   | 46     | $3.45 \times 10^{-5}$ | 0.27   | 859    | $5.87 \times 10^{-9}$ | 0.0658 |
| 10                  | 74.5   | $2.062 \times 10^{-6}$ | 0.338  | 794.5  | $2.47 \times 10^{-9}$ | 0.0395 |

But $K'_2 = G_2 \times \sqrt{D_2'}$ increases slowly as $D_2'$ decreases.

Evolved using metric $f_2$:

| Number of variables | $G_2$  | $D'_2$   | $K'_2$  |
|---------------------|--------|----------|--------|
| 2                   | 62.5   | $4.31 \times 10^{-3}$ | 4.1    |
| 3                   | 82.5   | $2.03 \times 10^{-3}$ | 3.71   |
| 4                   | 175    | $1.111 \times 10^{-3}$ | 5.83   |
| 5                   | 129.5  | $6.95 \times 10^{-4}$ | 3.41   |
| 6                   | 212.5  | $4.68 \times 10^{-4}$ | 4.6    |
| 7                   | 239    | $2.86 \times 10^{-4}$ | 4.04   |
| 8                   | 458    | $2.21 \times 10^{-4}$ | 6.81   |
| 9                   | 720    | $1.594 \times 10^{-4}$ | 9.09   |
| 10                  | 462.5  | $1.244 \times 10^{-4}$ | 5.16   |

A second try with the same parameters:

| Number of variables | $G_2$  | $D'_2$   | $K'_2$  |
|---------------------|--------|----------|--------|
| 2                   | 72.5   | $4.31 \times 10^{-3}$ | 4.76   |
| 3                   | 92.5   | $2.03 \times 10^{-3}$ | 4.17   |
| 4                   | 84.5   | $1.111 \times 10^{-3}$ | 2.82   |
| 5                   | 228.5  | $6.95 \times 10^{-4}$ | 6.02   |
| 6                   | 272.5  | $4.68 \times 10^{-4}$ | 5.9    |
| 7                   | 349.5  | $2.86 \times 10^{-4}$ | 5.91   |
| 8                   | 346.5  | $2.21 \times 10^{-4}$ | 5.15   |
| 9                   | 337.5  | $1.594 \times 10^{-4}$ | 4.26   |
| 10                  | 463    | $1.244 \times 10^{-4}$ | 5.16   |
Evolved using metric $f_3$:

| Number of variables | $G_2$   | $D_2$         | $K_2'$ |
|---------------------|---------|---------------|--------|
| 2                   | 60.5    | $4.31 \times 10^{-3}$ | 3.97   |
| 3                   | 107.5   | $2.03 \times 10^{-3}$ | 4.84   |
| 4                   | 255     | $1.111 \times 10^{-3}$ | 8.5    |
| 5                   | 203.5   | $6.95 \times 10^{-4}$ | 5.36   |
| 6                   | 410.5   | $4.68 \times 10^{-4}$ | 8.88   |
| 7                   | 470     | $2.86 \times 10^{-4}$ | 7.94   |
| 8                   | 613.5   | $2.21 \times 10^{-4}$ | 9.13   |
| 9                   | 718.5   | $1.594 \times 10^{-4}$ | 9.07   |
| 10                  | 863     | $1.244 \times 10^{-4}$ | 9.63   |

A second try with the same parameters:

| Number of variables | $G_2$   | $D_2$         | $K_2'$ |
|---------------------|---------|---------------|--------|
| 2                   | 69.5    | $4.31 \times 10^{-3}$ | 4.56   |
| 3                   | 105     | $2.03 \times 10^{-3}$ | 4.73   |
| 4                   | 152.5   | $1.111 \times 10^{-3}$ | 5.08   |
| 5                   | 292     | $6.95 \times 10^{-4}$ | 7.7    |
| 6                   | 196     | $4.68 \times 10^{-4}$ | 4.24   |
| 7                   | 457.5   | $2.86 \times 10^{-4}$ | 7.73   |
| 8                   | 379.5   | $2.21 \times 10^{-4}$ | 5.65   |
| 9                   | 859     | $1.594 \times 10^{-4}$ | 10.8   |
| 10                  | 794.5   | $1.244 \times 10^{-4}$ | 8.86   |

5 CONCLUSIONS

Evolving the second feature (with metric $f_2$ or $f_3$) always took more generations than the first feature (with metric $f_1$). At best, it took 8 times as many generations. Evolving a new feature into an already-working program is not easy; it is easier to evolve the new feature as a separate program. That is, evolving sorting in descending order is as easy as evolving sorting in ascending order. But evolving sorting in descending order while preserving sorting in ascending order is much harder. It’s easier to evolve something when you don’t have to keep something else working.

$D_2' \times D_1 < D_2$ (slightly). That is, programs that work according to $f_2$ are somewhat more abundant among programs that work according to $f_1$ than one would expect merely from knowing that all programs that work according to $f_2$ also work according to $f_1$.

The relationship $G_2 \sim \frac{1}{\sqrt{D_2}}$ approximately holds when evolving a second feature within a population of programs that implement a related first feature.
6  FURTHER QUESTIONS

What is the proportionality “constant”? (It’s not really constant, since it varies with population size, and maybe with other parameters.)

References

[1] Mike Stimpson [http://arxiv.org/abs/1102.2559](http://arxiv.org/abs/1102.2559) (2011)