Experiments with Cascading Design

Ben Kovitz

Fluid Analogies Research Group
Indiana University
512 N. Fess Av.
Bloomington, Indiana 47408 USA
bkovitz@indiana.edu

Abstract. Can interaction ("epistasis") within a genotype make a population "learn" to evolve faster? Two experiments described herein show that when fitness functions reward coordination among the parts of a phenotype, a genotype with a "cascading design"—one that produces a phenotype through cascades of simple elements triggering other simple elements—can exploit non-locality in the genotype-phenotype mapping to favor mutations that maintain valued coordinations. Cascading designs tend to evolve leverage points that increase the value of single mutations, enabling them to move quickly through a genotype space, skipping over valleys in the fitness landscape as they move from peak to peak.

1 Introduction

1.1 The problem of coordination

Most natural evolutionary processes successfully overcome an obstacle that frustrates most artificial evolutionary algorithms: greater fitness usually requires more-complex *coordination* among many traits. As individuals become more fit, changing one trait tends to destroy fitness unless all the other traits are changed appropriately to maintain the coordination among the traits.

Here is a simple example of coordination and how it presents obstacles to evolvability. In any good house design, the plumbing and electrical wiring connect water and electricity to the devices that need them. If you change the architectural plans for a house to move the bathroom from the northwest corner to the middle of the east wall, moving the walls and fixtures is not enough. You must also reroute all the pipes and electrical connections. There may also be further implications, such as required changes to the structure of the house, placement of doorways in other rooms, placement of windows, etc. As the design for a house becomes more refined, each new high-level change, such as where to place a whole room, becomes dominated more by what system engineers call "ripple": additional changes required to maintain integrity.

If an evolutionary algorithm is designing the house, and the genotype-phenotype mapping has a simple relationship between a change to each gene and a change to a correspending element of the house design, then moving the bathroom across the house requires that many mutations occur simultaneously: one mutation for each segment of pipe that needs to move, one mutation for each doorway, etc. As coordination becomes more complex, the probability of making all the needed mutations simultaneously gets lower and lower.

Ripple is especially troublesome for practical evolutionary algorithms because usually a design that lacks the needed coordination is completely worthless. For example, plumbing that doesn't connect to the water supply simply doesn't work at all. A vertebrate whose circulatory system wasn't connected to its heart, or that didn't complete a full circuit back to the heart, could not survive at all. Thus, the need for coordination induces large "valleys" in fitness landscapes: regions that cannot be crossed by a sequence of small mutations, because any one mutation that puts an individual into the valley ensures that the individual will not reproduce at all. As noted in Gavrilets (1997), the fitness landscapes of real organisms tend to be dominated by regions where nothing is viable.

Yet biological evolution seems to maintain coordination among an organism's traits with (relative) ease. In fact, biological evolution seems to have gotten faster and faster as organisms have become more complex. For example, Benton and Emerson (2007) found that the number of families of mammals has been increasing exponentially since the K-T boundary. This suggests that it has been becoming progressively easier for mutations to find new, beneficial, radical changes. A simple piece of evidence is just the fact that it took 2 billion years of mostly single-celled organisms to reach the Cambrian explosion, but in the last 542 million years, more-complex forms of life have been varying much more radically than Precambrian life ever did. In biology, at least, complexity seems to favor evolvability.

Crossover enables some degree of coordinated change. Two house designs with the bathroom in different places can "cross", resulting in a design where the bathroom from one house is superimposed on the design of the other house. The resulting design might even have connected plumbing and electrical systems. But crossover includes no mechanism to preserve relationships between traits separately from the absolute measurements of those traits. Its main benefit is to combine two internally coordinated subsystems without destroying either one.

1.2 Cascading design

This paper describes some experiments that explore the hypothesis that *cascading designs* become progressively more evolvable against fitness landscapes that reward coordination among traits. This might explain why natural evolution is so successful at evolving organisms with such complex forms of coordination.

A cascading design is a network of elements that each do very little on their own, mainly triggering other elements, possibly adding some small twist of their own to whatever triggered them. The classic example of a cascading design is a metabolic network: each enzyme does little more than catalyze reactions that create or consume other enzymes or speed up or slow down their activity.

Hypothesis: When fitness rewards coordination (invariant relationships among the traits or parts of a phenotype), and a genotype is a cascading design that can alter its structure as well as the quantitative parameters within it, the genotype will respond to selective pressure to favor structures that are *better at evolving.*

The argument for the plausibility of this hypothesis is that a cascading design can evolve "leverage points" where a single mutation will create a coordinated change. A change to an element early in a cascade propagates through elements later in the cascade, eventually spreading the change throughout the phenotype. The network through which the cascade propagates thus represents relationships within the phenotype that are preserved even as the single mutation alters the whole phenotype.

The experiments in this paper test this hypothesis through evolutionary algorithms in which a fitness function varies randomly every 20 generations, but within some constraint so that each fitness function rewards the same type of coordination even as it changes the absolute values that describe an optimal phenotype. Each time the fitness function changes, we should see a "race" toward the new optimum: the winners of the race should have genotypes whose structure makes them faster at moving up the gradient of the fitness function. Invariant forms of coordination imply some sort of common type of gradient in all the fitness landscapes, which a well-tuned cascading structure should be able to exploit.

2 Related work

There is a great deal of work, starting with Kauffman (1969), on NK models: random networks of N nodes, where the fitness of the network is the sum of the fitnesses of each node, and each node's fitness is a function of a number stored at K neighboring nodes. The main result is that when K=1, the fitness landscape is hill-shaped, and as K increases (more interactions within the network), the fitness landscape becomes more "rugged": small changes to the network result in large, uncorrelated changes to fitness. Smith et al. (2002) analytically derived the probability that a mutant will have greater fitness than its parent. As K increases, the probability increases that a parent with above-average fitness will produce a mutant with lower fitness. When K is high, the average fitness of the population is likely to progress more slowly at higher levels of fitness: the lack of correlation between nearby genotypes and nearby fitness levels means that a small change to a high-fitness genotype is more likely to lead to a lower-fitness genotype.

Most measures of evolvability have focused on the probability of improvement in a single generation, deducing from a "transmission function" specifying the probability that parents with a specified genotype and phenotype will produce a child with a specified genotype and phenotype (Cavalli-Sforza and Feldman, 1976). "Evolution of evolvability" means evolution that increases the expected fitness of a mutant. Altenberg (1994) describes a number of techniques to achieve this, such as genes that affect the probability that other genes will be mutated, and the evolution of subroutines in genetic programming.

It is generally held that better evolvability is promoted by high "locality" in the mapping from genotype to phenotype: that is, a small change in the genotype should produce a small change in the phenotype (Rothlauf and Oetzel, 2006), with the exception that low locality makes it easier to escape from a local optimum.

Metabolic networks have been found to be modular: they consist of clusters of interacting enzymes with relatively few interactions with enzymes in other clusters. Similarly for genomes. It is not known whether modularity evolves because of selective pressures or mutational biases (Wagner et al., 2007). Clune et al. (2013) propose that connections are themselves costly to an organism, creating a selective pressure to keep connections to a minimum. Kashtan and Alon (2005) find that changing the fitness function tends to cause the creation of modules in genotypes produced by standard evolutionary algorithms.

Kirsten and Hogeweg (2011) found *in silico* that a developmental difference that makes mutations tend to preserve body structure also led to better evolvability. Hogeweg (2012) explores the tendency of genotypes to integrate information into their structure over time, producing "non-random" (that is, biased) exploration of the genotype space and increasing evolvability. The present paper explores the same line of thought, mathematically simplified and abstracted further from biology—a further step toward isolating this aspect of evolvability and finding a rigorous and general theory to explain it.

3 Data and methods

All experiments consisted of running an evolutionary algorithm implemented in the Scala programming language. In each experiment, the fitness function operated on "phenotypes", which are defined to be vectors of four real numbers.

Each experiment had two parts: "direct evolution" and "cascading design". In direct evolution, the population consists of phenotypes; or in other words, the genotypes and the phenotypes are identical, so variation operates by directly modifying the phenotypes. In cascading design, the genotypes consist of directed graphs, which map to phenotypes as described in section 3.1.

The fitness function varies once per epoch. The same fitness function is always applied in the same epochs to both the direct and cascading-design populations.

3.1 Evolutionary algorithm

In the first epoch of each experiment, generation 1 contains randomly generated individuals. All succeeding epochs start with a "generation 0", which contains the same individuals as the last generation of the previous epoch.

Each succeeding generation is made by varying individuals from the preceding generation. There are two types of variation: mutation and crossover, described below. Individuals are chosen for variation (one individual for mutation, two individuals for crossover) by tournament selection with $n = 3$: three individuals are randomly selected, and the one with the best fitness is chosen. No two instances of the same individual are ever allowed in the same population.

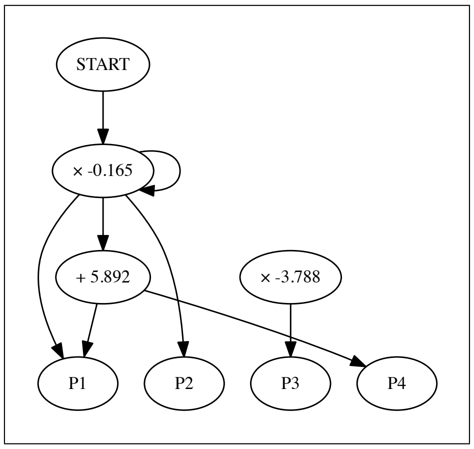

Fig. 1. A directed graph representing a simple cascading design.

The cascading-design genotype Each genotype in a cascading-design population is a directed graph containing the following:

1. A single `START` node.
2. A set of "phenotype nodes" `P1`...`P4`, each corresponding to an element of the phenotype vector.
3. A set of "genotype nodes" `G1`...`G`n, each associated with a *weight* and an *operation.* The weight is a real number in the range [–10.0, +10.0]. The operation is either + or ×. There can be any number of genotype nodes, zero to infinity, and they do not need to be numbered contiguously.
4. A set of directed edges from one node to another. The `START` node takes no incoming edges. The phenotype nodes do not have outgoing edges. Otherwise, for any two nodes a and b, there is either one edge from a to b, or there is no edge from a to b.

A simple directed-graph genotype is shown in figure 1.

Genotype–phenotype mapping A directed graph as in the preceding section maps to a phenotype vector of length n according to the following algorithm, in which each node is assigned an activation level (a real number), or simply an "activation" for short.

1. Let the `START` node have an activation of –10.0.
2. For each phenotype `P`i, select the subgraph containing only the paths from `START` to `P`i. In the remaining steps, the only nodes and edges considered are those within this subgraph.
3. For each genotype node `G`j, assign it an activation equal to its operation and weight applied to the sum of the activations of all of its inputs. For example, if `G`j has operation × and weight 2.0, and two incoming edges from nodes having

activations 1.0 and 2.0, then assign Gj an activation of $2.0 \times (1.0 + 2.0) = 6.0$. If G$j$ has no inputs with activations, then do not assign it an activation.

4. When every phenotype node has an activation, assign Pi an activation equal to the sum of the activations of its inputs, and continue with the next phenotype node at step 2.

5. When all the P nodes have an activation, this yields the resulting phenotype: the ith element of the phenotype equals the activation of Pi. If there is no path from START to a given node Pi, then the corresponding element of the phenotype does not get any number at all. Any part of a fitness function for which that phenotype element is an argument, will contribute 0.0 to the total fitness (see the experiments, below, for more about how the various fitness functions are calculated).

The directed graph thus defines a set of single-argument functions, composed of the simple functions specified by each genotype node. Applying each of these functions to the argument –10.0 yields the phenotype; or in symbols, $x_i = f_i(-10.0)$ where x_i is the ith element of the phenotype.

A fully worked example will clarify the process, especially regarding the way cycles in the graph affect the results. The four tables below show each iteration involved in calculating the activation of each of the four P nodes in the directed graph of figure 1.

iteration	START	$\times -0.165$	$+5.892$	P1
1	-10.0	none	none	none
2	-10.0	1.650	none	none
3	-10.0	3.028	7.542	10.570

Note that the $\times -0.165$ counted into the final sum for P1 twice because of the cycle and because once it had an activation, it took two iterations to get activations in all the genotype nodes.

iteration	START	$\times -0.165$	P2
1	-10.0	none	none
2	-10.0	1.650	1.650

This time, $\times -0.165$ only counted into the final sum once, because $+5.892$ is not on the path from START to P2.

iteration	START	P3
1	-10.0	none

The node $\times -3.788$ does not count toward P3 at all, because $\times -3.788$ is not on the path from START to P3. Since there is no path at all from START to P3, P3 gets no activation at all.

iteration	START	$\times -0.165$	$+5.892$	P4
1	-10.0	none	none	none
2	-10.0	1.650	none	none
3	-10.0	3.028	7.542	7.542

Note that even though node $\times -0.165$ was assigned a new activation on the last iteration, as in the calculation for P1, the new activation did not figure into the activation for `P4`. This is because `P4` only has one input, and only the current activation counts.

Mutation For direct evolution, mutation is simple: a randomly chosen element of the phenotype has a random number in the range [–0.2, +0.2] added to it. The random number is uniformly distributed.

For cascading design, there are six possible mutations, shown in Table 1. The probabilities are chosen to make it easier to exploit an existing cascade than to make a new one. If the hypothesis is correct, it should be possible to evolve very efficiently by making small quantitative adjustments—turning "knobs"—than by modifying the structure of the genotype. But mutations to modify the structure of the genotype are still possible, so populations can respond to selective pressure favoring one kind of structure over another.

Because each mutation in direct evolution changes the phenotype only a small amount, the number of mutations made to an individual in direct evolution is randomly chosen to be from 1 to the number of elements in the phenotype (uniformly distributed). In cascading design, however, only a single mutation separates a parent from each of its offspring.

Table 1. Mutations to cascading designs and their probability weights.

Weight	Name	Description
8	jiggleRandomNode	Add uniformly distributed random number in $[-0.2, +0.2]$ to randomly chosen genotype node.
4	moveEdge	Move an edge from one place in the graph to another.
1	addEdge	Add an edge where one does not already exist.
1	removeEdge	Remove an existing edge.
1	addNode	Add a new node and give it one incoming edge and one outgoing edge to connect it with other, existing nodes. Choose $+$ or $\times$ randomly and assign weight randomly, with uniform distribution.
1	removeNode	Remove a randomly chosen genotype node.

Crossover For direct evolution, crossover consists of choosing each phenotype element with 50% probability from each of the parents.

For cascading design, crossover works as follows. For each genotype id (`G1`, `G2`, etc.), the offspring gets a copy with equal probability from either parent. All

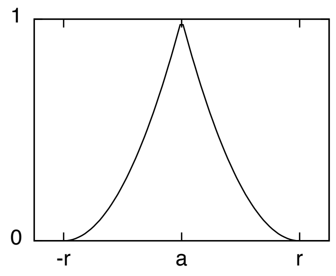

Fig. 2. The $\Lambda(x, a; r)$ function rewards matching a target number, but only within a radius of sensitivity r.

edges from the parents are copied to the offspring, subject to the restriction that their source and destination nodes must also exist in the offspring.

The Λ function Several fitness functions in the experiments below make use of a special "inverted-U" function, plotted in figure 2. This function is defined as:

$$\Lambda(x, a; r) = \begin{cases} 0 & \text{if } |x - a| > r \\ \left(\frac{r - |x-a|}{r}\right)^2 & \text{if } |x - a| \leq r \end{cases}$$

where x is the number to be tested, a is a target number, and r is a radius of sensitivity. Since the plot of the function looks more like an inverted V than an inverted U, it has the name Λ. It rewards an exact match of the target, falls away sharply for inexact matches, and gives no reward at all for numbers further than r away from the target.

3.2 Experiment 1: Coordination gateway

In the first experiment, populations are subjected to a fitness function with a "gateway": if one condition is met (or nearly met), then points are awarded for meeting the second condition. If the first condition is not met (or nearly met), then no points are awarded for the second condition, even if the individual meets that condition. Parameters affecting the absolute values needed to meet the second condition change from epoch to epoch, but the first condition specifies only a ratio and a range for the first two elements of the phenotype vector. Each phenotype vector is four elements long.

The purpose of a gateway function is to simulate, in the simplest possible way, an organism's need to maintain a certain relation between some of its traits while it seeks optimal absolute values for those traits as well as others. To take a cartoonish example, it helps if your legs are the same length. Changing environments may reward longer or shorter legs, but legs of unequal length frustrate coordination so much that it's impossible to make use of your other capabilities to thrive, regardless of the environment. But having equal-length legs is certainly not enough: your other traits—tooth size, intestine length, etc.—also need to be able to evolve in response to selective pressure, without undermining equal leg length.

Suppose that the environment rewards longer legs than the organism has. A mutation that lengthens one leg without changing the length of the other would be strongly selected *against*, because unequal leg length is strongly penalized. Since such an organism would likely not be viable, there would be no opportunity in a future generation to get a second mutation, and therefore no opportunity to bring the other leg up to the same length as the first.

So, organisms that are able to change the lengths of both legs equally by a single mutation will be much more evolvable than organisms that organisms whose genomes are structured so that changing the length of each leg requires a separate mutation. These organisms are always generating offspring with longer legs as well as offspring with shorter legs, so, as the environment changes to sometimes favor longer, sometimes shorter legs, some of the offspring will have an advantage. These organisms will also be able to take advantage of other variations in the environment as they come, if their genomes are set up to vary other traits while leaving leg length unchanged.

Number of runs: 80; epochs per run: 40; generations per epoch: 20; population size: 40; mutation/crossover: 90%/10%.

Fitness function The exact fitness function applied in this experiment is as follows. Each epoch, two constants, c_{12} and c_{34}, are randomly chosen in the range [+1.0, +10.0]. The total fitness is calculated from two partial fitness functions, w_1 and w_2, explained below.

$$\begin{aligned} w_1 = {} & \Lambda(|x_1 - x_2|, 0; 0.1) + \\ & s(x_1 - 1) + s(x_2 - 1) \end{aligned}$$

where x_1 and x_2 are the first two elements of the phenotype, Λ is the function defined in section 3.1, and s is a sigmoid function with slope of 10.0 at $x = 0$. Thus w_1 is a measure of how closely x_1 and x_2 equal each other. Since the radius of sensitivity is 1.0, no points are awarded for equality if $|x_1 - x_2| > 0.1$. Each s function awards up to 1 point for the phenotype element being greater than 1.0. So, w_1 falls in the range [0.0, 3.0].

$$\begin{aligned} w_2 = {} & 10\Lambda(|x_1 - c_{12}|, 0; 6.0) + \\ & 10\Lambda(|x_2 - c_{12}|, 0; 6.0) + \\ & 10\Lambda(|x_3 - (c_{34} - 2)|, 0; 6.0) + \\ & 10\Lambda(|x_4 - c_{34}|, 0; 6.0) \end{aligned}$$

In words, w_2 rewards up to 10 points for each of x_1 and x_2 approximating c_{12} and up to 10 points for each of x_3 and x_4 approximating c_{34}, except that the target for x_3 is always 2 less than x_4. The radius on the Λ function is 6.0, so even very inaccurate values are rewarded.

The total fitness is:

$$w = \Lambda(w_1, 3.0; 1.0) \cdot w_2$$

So, the total fitness is between 0.0 and w_2, depending on how close w_1 is to its maximum. Because the radius of sensitivity is 1.0, if $w_1 < 2.0$ then $w = 0$.

3.3 Experiment 2: Invariant ratio

In this experiment, a target phenotype is randomly chosen each epoch, with the restriction that the ratio between each element and the next is always 2. For example, a possible target phenotype in this experiment is $[1.07, 2.14, 4.28, 8.56]$.

The closeness of a given element of the phenotype is measured by the Λ function:

$$w = \sum_i \Lambda(x_i, y_i; 2.0)$$

where x_i is the ith element of the phenotype and y_i is the ith element of the target. So, fitnesses are always within $[0.0, 4.0]$.

The fourth element of the target phenotype, y_4, is always chosen to fall within the range $[5.0, 10.0]$. This ensures that the first two elements aren't indistinguishably close to zero.

Number of runs: 80; epochs per run: 40; generations per epoch: 20; population size: 40; mutation/crossover: 100%/0%.

4 Results

4.1 Experiment 1: Coordination gateway

As figure 3 shows, the best fitnesses of the last generation of each epoch progressively increased for roughly the first 12 epochs. The variance in the best fitness decreased up to epoch 40.

As expected, direct evolution did not improve from epoch to epoch. Direct evolution often reached the last generation of an epoch with a best fitness of 0.0, because it never got the first two phenotype elements to equal ("both legs the same length"). Consequently, it often saw little or no fitness gradient. When it did happen upon a phenotype with $x_1 \approx x_2$, it was able to hill-climb, but not very quickly or steadily. Many generations had a lower best fitness than the previous generation.

After a few epochs, cascading design was able to consistently "pass through" the gateway and hill-climb x_1, x_2 without destroying the relationship between them, and also hill-climb x_3, x_4 without undermining x_1, x_2. For cascading design, the first generation of nearly every epoch (after the first few) *began* mostly with graphs that made phenotypes with $x_1 = x_2$. So, cascading design started in the sweet spot of the fitness landscape, which otherwise must be searched for without benefit of a gradient. For the first few epochs of each run, though, cascading design often produced graphs with zero fitness in every generation. But once a cascading-design population found graphs that generated $x_1 = x_2$,

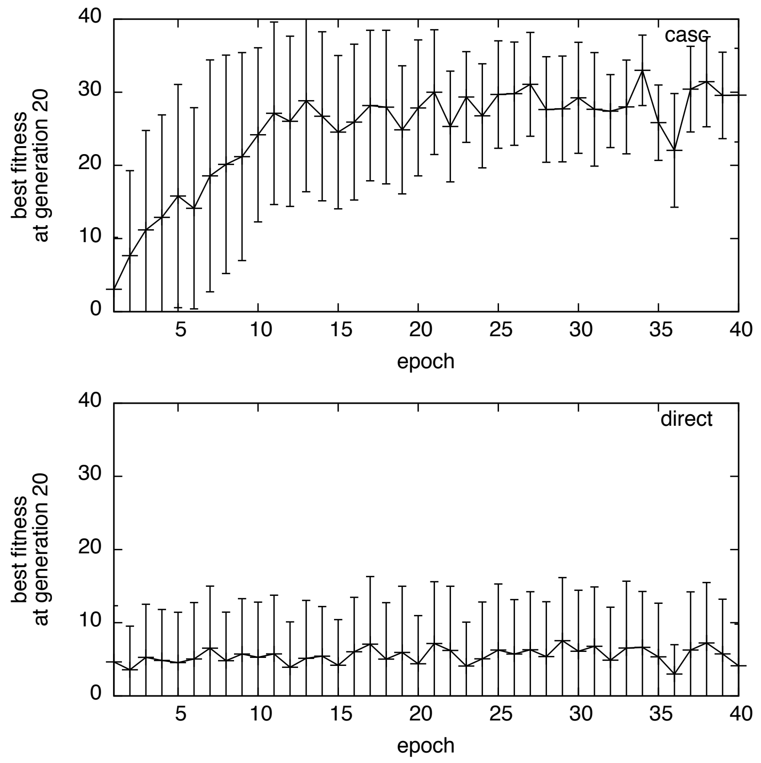

Fig. 3. Best fitnesses in the last generation of each epoch of the "coordination gateway" experiment. Error bars show one standard deviation above and below the fitness.

most populations from that point on consisted mostly of graphs that preserved that property. Very few generations had a best fitness worse than the previous one.

Figure 4 shows a directed graph from epoch 38, generation 20 of one run, when the fitness function had c_{12}=5.554, c_{34}=8.572. It generates the phenotype [5.496, 5.496, 6.586, 8.804] and has a fitness of 38.813, the best of its generation.

Note that since `P1` and `P2` both take their only input from the same node, varying any number in the genotype will leave `P1` and `P2` equal. Most genotypes that evolve after a few epochs in the "coordination gateway" experiment have this property. Most beneficial mutations in this experiment varied all four phenotype elements but left `P1` and `P2` equal.

Note also that varying the node $\times$`-5.701` adjusts the relationship between `P3` and `P4`, while varying any of the four genotype nodes above it modifies all four phenotype nodes but preserves the relationship between `P3` and `P4`.

The more-complex graph in figure 5 has similar properties. This occurred in epoch 15, generation 11 of the same run. In that epoch, the fitness function had

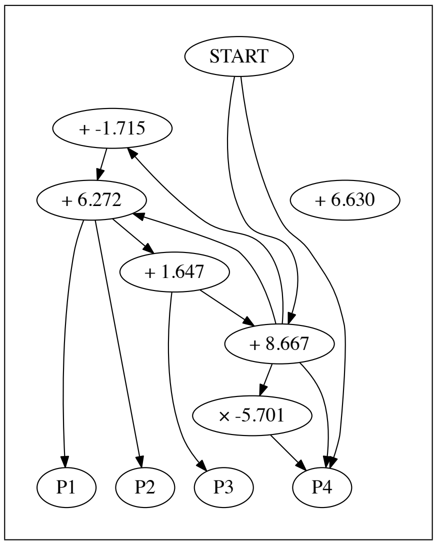

Fig. 4. A directed graph from the "coordination gateway" experiment. Varying any number leaves `P1` = `P2`.

c_{12}=1.077, c_{34}=3.818. The resulting phenotype is [13.790, 13.790, 1.649, –3.349], with a fitness of 9.444, 30th (out of 40) in its generation.

The graph in figure 5 illustrates a common feature of many graphs that evolved in both experiments: genotype nodes occur in chains or well-connected communities that provide many mutation targets, each of which varies the same "global" parameter of the phenotype nodes of direct relevance to the fitness function. Thus, the probability is very high that any mutation will preserve coordinations that have proven crucial in previous evolution.

Crossover tended to be valuable in early epochs, but in later epochs tended to produce catastrophic results. Cartesian Genetic Programming (CGP) often uses mutation exclusively, because crossover tends to be disruptive (Miller, 2011, pp. 29–30). CGP programs are cascading designs quite similar to the directed graphs used in the experiments described in this paper.

4.2 Experiment 2: Invariant ratio

Similarly to Experiment 1, cascading design became progressively better at evolving until leveling off around epoch 15; see figure 6. As expected, direct evolution did not improve from epoch to epoch.

The graphs generated by cascading design often had about twice as many nodes in them as in Experiment 1, but about half of the nodes were inactive; see figure 7. This graph comes from epoch 40, generation 20 of a run (the last generation of the last epoch), when the target phenotype was [1.049, 2.097, 4.194, 8.389]. It generates a phenotype of [1.145, 2.130, 3.892, 8.370], with a fitness of 3.575, the best of its generation.

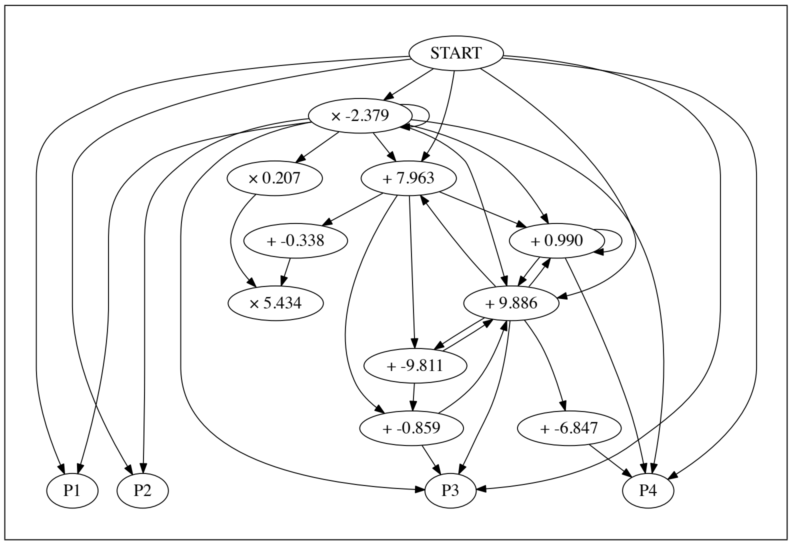

Fig. 5. A more-complex directed graph from the "coordination gateway" experiment.

This graph displays a structure typical of most successful graphs in the "invariant ratio" experiment: most genotype nodes affect all phenotype nodes simultaneously, but the number of interacting genotype nodes tends to increase a little bit from left to right.

The cluster of highly connected active genotype nodes is easy to explain: it embodies the invariant ratio of 2.0 between successive elements in all target phenotypes. Changing a genotype node will tend to alter all four phenotype nodes while preserving the ratios between them.

5 Conclusions

The results establish that cascading designs respond to an indirect selective pressure to favor structures that evolve faster, when exposed to a variety of fitness functions that reward an invariant kind of coordination in the phenotype.

In effect, the structures of the genotypes, as opposed to the numbers in the genotype nodes, serve as a memory of previous fitness functions. The mathematical structure of the fitness-function space leaves a mark on the structure of the surviving genotypes, distinct from the constants that vary from one fitness function to the next.

Contrary to the usual difficulties with non-locality in a genotype–phenotype mapping, the cascading designs *exploited* non-locality to move more quickly and more directly through the phenotype space than is possible by hill-climbing the phenotypes directly.

Contrary to the behavior usually found in *NK* models, the fitness landscape appeared to become *less* rugged for the cascading designs.

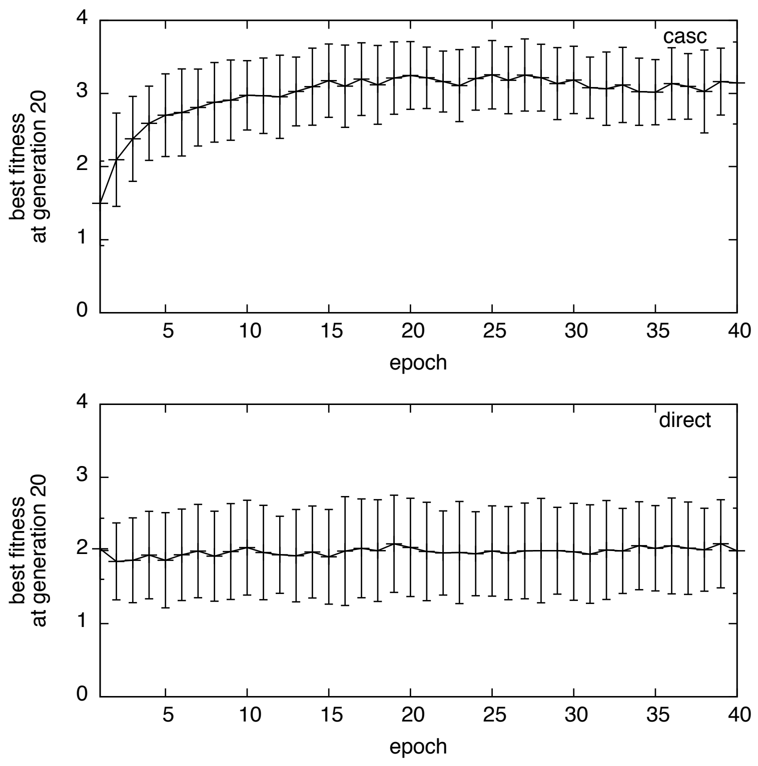

Fig. 6. Best fitnesses in the last generation of each epoch of the "invariant ratio" experiment. Error bars show one standard deviation above and below the fitness.

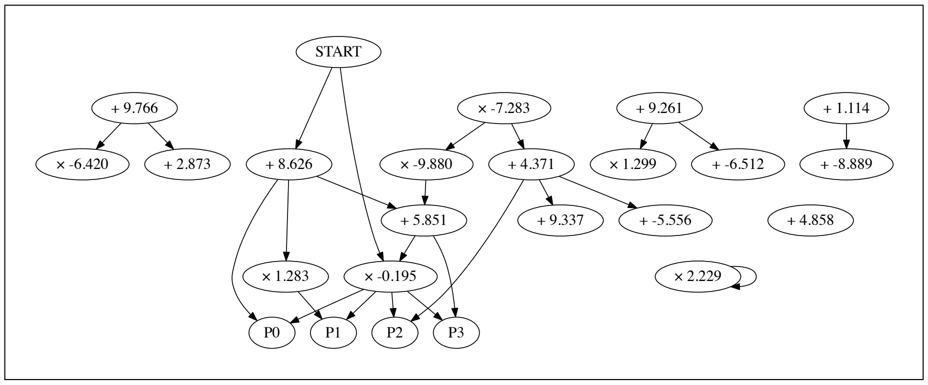

Fig. 7. A directed graph from the "invariant ratio" experiment.

Bibliography

Altenberg, L. (1994). The evolution of evolvability in genetic programming. *Advances in genetic programming 3*, 47–74.

Benton, M. J. and B. C. Emerson (2007). How did life become so diverse? the dynamics of diversification according to the fossil record and molecular phylogenetics. *Palaeontology 50*(1), 23–40.

Cavalli-Sforza, L. L. and M. W. Feldman (1976). Evolution of continuous variation: Direct approach through joint distribution of genotypes and phenotypes. *Proceedings of the National Academy of Sciences 73*(5), 1689–1692.

Clune, J., J.-B. Mouret, and H. Lipson (2013). The evolutionary origins of modularity. *Proceedings of the Royal Society b: Biological sciences 280*(1755), 20122863.

Gavrilets, S. (1997). Evolution and speciation on holey adaptive landscapes. *Trends in ecology & evolution 12*(8), 307–312.

Hogeweg, P. (2012). Toward a theory of multilevel evolution: long-term information integration shapes the mutational landscape and enhances evolvability. In *Evolutionary Systems Biology*, pp. 195–224. Springer.

Kashtan, N. and U. Alon (2005). Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences of the United States of America 102*(39), 13773–13778.

Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology 22*(3), 437–467.

Kirsten, H. and P. Hogeweg (2011). Evolution of networks for body plan patterning; interplay of modularity, robustness and evolvability. *PLoS computational biology 7*(10), e1002208.

Miller, J. F. (2011). *Cartesian genetic programming.* Springer.

Rothlauf, F. and M. Oetzel (2006). *On the locality of grammatical evolution.* Springer.

Smith, T., P. Husbands, P. Layzell, and M. O'Shea (2002, March). Fitness landscapes and evolvability. *Evol. Comput. 10*(1), 1–34.

Wagner, G. P., M. Pavlicev, and J. M. Cheverud (2007). The road to modularity. *Nature Reviews Genetics 8*(12), 921–931.