Heterochronic Scaling of Developmental Durations in Evolved Soft Robots

John Rieffel
Union College
Schenectady, NY 12308 USA
rieffelj@union.edu

ABSTRACT

In the evolution of Generative and Developmental Systems (GDS), the choice of where along the *ontogenic trajectory* to stop development in order to measure fitness can have a profound effect upon the emergent solutions. After illustrating the complexities of ontogenic fitness trajectories, we introduce a GDS encoding without an *a priori* fixed developmental duration, which instead slowly increases the duration over the course of evolution. Applied to a soft robotic locomotion task, we demonstrate how this approach can not only retain the well known advantages of developmental encodings, but also be more efficient and arrive at more parsimonious solutions than approaches with static developmental time frames.

Categories and Subject Descriptors

I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Robotics*

General Terms

Algorithms, Design, Experimentation

Keywords

Soft Robot, Generative and Developmental System, Simulation

1. INTRODUCTION

One of the more significant advances in the understanding of biological evolution has been the emerging field of Evolutionary Developmental Biology (Evo-Devo), which highlights the role of development (*ontogeny*) as an agent of evolutionary change. Lately, ideas of Evo-Devo have been integrated into evolutionary search as well, with a number of positive results.

There are a range of approaches which all fall under the rubric of Generative and Developmental Systems (GDS), including L-System grammars [20], Genetic Regulatory Networks [3], and HyperNEAT [27]. These developmental systems have been used to design everything from robots [25, 4, 6, 16, 19, 17] to satellite antenna [18].

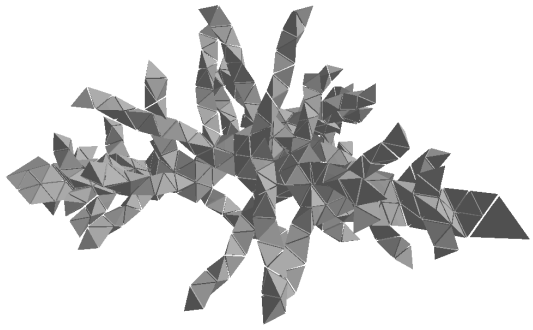

Figure 1: Soft tethrahedral mesh robots like this one can be grown and evolved using developmental encodings. This paper explores how to scale the length of the developmental process in order to grow more complex shapes.

What distinguishes developmental systems from the conventional Genetic Algorithm is the *prescriptive* rather than *descriptive* representation of solutions. In the conventional GA there is no distinction between genotype (the unit of variation) and phenotype (the unit of selection), and therefore the genotype fills both roles. In Generative and Developmental Systems, by contrast, a genotype describes a *process* of development which *results* in the phenotype. Consequently, small changes in a genotype can lead to large and co-ordinated changes in the phenotype. Regardless of implementation, developmental encodings all offer many of the same benefits and advantages, among them implicit modularity, symmetry, and large-scale co-ordinated changes [16, 15]

One consequence of the prescriptive nature of developmental systems is that there is no longer a single phenotype with which to measure fitness. Rather, ontogeny creates an entire progression of phenotypes, from initial "seed" to fully formed "adult", each with some intrinsic fitness. Gould, a major contributer to Evo-Devo, refers to this progression as an *ontogenic trajectory* [10]. From a biological perspective, small changes in ontogenic trajectories (*heterochrony*) can cause correspondingly large changes in phenotypes [22]. A famous example is the water-based *axolotl* salamander,

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

which as an adult resembles the larval stage of the land-based species it is believed to have descended from [21].

The issue of developmental timing, in particular *where* along the ontogenic trajectory to terminate development in order to evaluate the resulting phenotype is a complicated one, and relatively unexplored. Lately, a few papers have explored more adaptive approaches to developmenal timing and have described consequential advantages over schemes with static endpoints, including improved robustness [8] and generality [14] of solutions, among others [28, 9].

The main goal of this paper is to further explore the link between ontogenic trajectories, their endpoints, and the qualities of the emergent solutions. After a review of approaches to developmental timing in related papers, we introduce our own developmental system - a face encoding grammar which grows tetrahedral soft meshes. We use this grammar to explore the topology of ontogenic trajectories, indicating some of the underlying pathologies of fixed ontogenic endpoints. We then introduce a small change to the developmental system which slowly increases the duration of ontogeny over the course of evolution. The impact of this developmental scaling technique is experimentally studied on a soft robotic locomotion problem. Comparing this scheme to the fixed-timing approach, we show how this it can be more efficient and arrives at more parsimonious solutions than the fixed model.

2. ONTOGENIC TRAJECTORIES AND ENDPOINTS

Almost all developmental systems are based around an iterative process by which a small "seed" slowly develops into increasingly complex (one hopes) phenotypes. For instance, in the case of grammars operating on strings, such as L-Systems [20], the "seed" is the initial starting string, and development proceeds iteratively as successive rewrite rules are applied to the growing string. In Genetic Regulatory Networks, genes are iteratively expressed in order for processes such as cell division and growth to occur [5].

Developmental processes therefore create an entire progression of genetically identical but phenotypically distinct intermediate solutions, and each of these solutions has, in principle, its own intrinsic fitness. This landscape of intermediate phenotypic fitnesses is referred to as an *ontogenic trajectory*. Since the process of development is largely open-ended, there is no one absolute end point for the ontogenic trajectory . Moreover, there is no guarantee that intermediate phenotypic fitness increases monotonically (or at all) over the course of growth. The choice of when to stop development in order to measure fitness – what we call the *ontogenic endpoint* is an incredibly important one. Devert *et al.* have aptly dubbed this choice the "halting problem" of developmental systems [8].

In an early critical analysis of developmental systems, Viswanathan [29] shows how a poor choice of ontogenic endpoints can have a deleterious effect upon the success of evolution. He proposes one way this can be overcome, by using an "informed genotype-phenotype map" which always returns the highest fitness along the trajectory (with some arbitrary upper bound). This informed G-P map is a deliberate straw-man – it is incredibly resource intensive, and hardly practical for problems of any reasonable level of complexity.

Many implementations of developmental systems therefore choose to set the ontogenic endpoint *a priori*, setting a fixed number of iterations for the developmental process. For instance, Bongard and Pfeifer's GRN-based evolution of virtual agents, limited their development to 300 [4] and 500 [6].

Others choose the ontogenic endpoint based upon phenotypic properties. Hornby's L-system based evolution of tables and robots, for instance, limited the resulting phenotype string to 10,000 rules [17, 16]. Bongard and Auerbach's structure [2] and creatures [1] grown with CPPNs terminated growth after the phenotypes reached a certain density or maximum number of particles.

Lately a few researchers have introduced various models of adapting, either implicitly or explicitly, the ontogenic trajectory, and several have demonstrated some advantages of these variable timings. Hoang *et al.* use what they call "Evolutionary Developmental Evaluation" [14], in which individuals are evaluated on different problems , for instance increasing expansions of a polynomial series at different stages of development, and show an improvement in the generality of evolved solutions. Chavoya and Duthen's cellular automata contain a genetically specified "control field" which codes for the number of steps the CA is run [7]. Devert *et al.*[8] use a stopping criterion based upon the stability of an internal energy model, and argue that this results in ontogenies which are more robust in the presence of noise. Steiner *et al.* evolve vector field embryogeny capable of encoding a time frame for development, and demonstrate that the resulting solutions avoid over-fitting relative to a non-adaptive model [28]. Federici and Downing [9] use a method which gradually adds developmental stages to a developmental process as evolution proceeds, and argue that this leads to more scalable solutions. Toward the end of their paper, they mention that they are able to cache earlier embryonic stages and therefore see a positive effect upon simulation speed.

Our aim in this paper is to provide a novel, and in some ways simpler, means of scaling ontogenic endpoints over the course of evolution.

3. A FACE ENCODING GRAMMAR TO GROW TETRAHEDRAL MESHES

In order to demonstrate the pathologies of ontogenic trajectories, and in order to describe our approach to the scaling of developmental durations, we must first introduce our developmental encoding. Our system is used to create tetrahedral meshes representing soft robot morphologies. Tetrahedra are defined by four vertices and four corresponding faces, as illustrated by the image in the top left of Figure 2. A mesh is formed by connecting adjacent tetrahedra at common vertices. Tetrahedral meshes are commonly used in Finite Element Analysis (FEA) as well as physics-based game engines, such as NVidia's PhysX, which we use in this paper.

The developmental encoding we employ, a face-encoding grammar, uses a sequence of rewrite rules which operate on the faces of these tetrahedra. Similar systems, operating on graph edges rather than tetrahedral faces, have been used to grow both 3-D surfaces[11] and large tensegrity structures [24].

Beginning with the idea that each exposed face of a tetrahedron can be given a label, we specify three operations

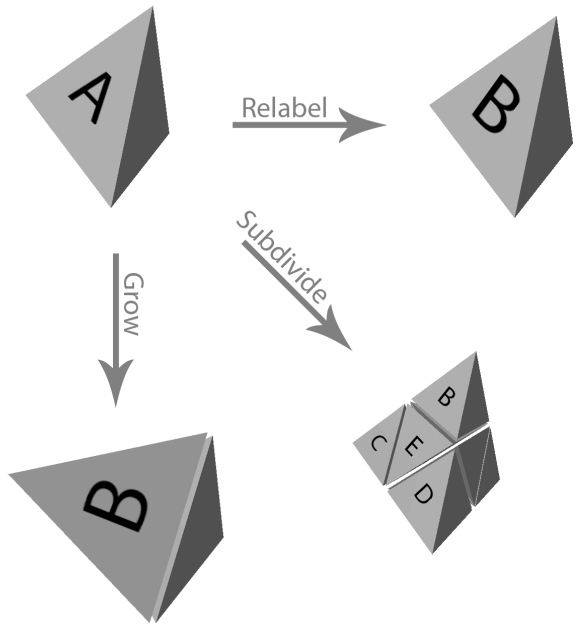

Figure 2: Our grammar specifies three rules which can be applied to the face of a tetrahedron. Clockwise from top left: the original tetrahedron with face labeled "A", relabel replaces "A" with "B", subdivide replaces the face with four smaller faces (this requires subdividing the entire tetrahedron), and grow adds a new tetrahedron with face labels "B","C","D"

which can be performed upon a face, as illustrated in Figure 2. For the sake of simplicity, we assert that these operators are only applied to *exposed* faces, – that is, those which are not shared by two adjacent tetrahedra.

> $A \rightarrow relabel(B)$ will replace a face labeled 'A' with one labeled 'B'
>
> $A \rightarrow grow\{BCD\}$ replaces a face labeled 'A' with a new tetrahedron, labeling the new exposed faces as 'B','C', and 'D'.
>
> $A \rightarrow divide[BCDE]$ subdivides a face 'A' into four smaller faces, 'B','C','D', and 'E'. The underlying tetrahedron must also be subdivided.

We also include a single "initialize" rule which is applied once to the faces of a starting "seed" tetrahedron

> $Initialize[BCDE]$ - applied once at the beginning of growth to establish the initial face labels of the seed tetrahedron.

A "ruleset" can then be created by specifying a fixed number of nonterminal labels, and providing a rewrite rule for each label. Table 1 shows one such ruleset.

We can now grow tetrahedral meshes of arbitrary size by iteratively applying a given ruleset to an initial "seed" tetrahedron. Figure 3 shows the growth of one such tetrahedral mesh.

$Initialize[AGCA]$			
A	$\rightarrow$	grow	$\{DBF\}$
B	$\rightarrow$	grow	$\{ADF\}$
C	$\rightarrow$	grow	$\{EDF\}$
D	$\rightarrow$	relabel	(D)
E	$\rightarrow$	grow	$\{DCF\}$
F	$\rightarrow$	divide	$[DDDG]$
G	$\rightarrow$	grow	$\{DDG\}$

Table 1: An example ruleset.

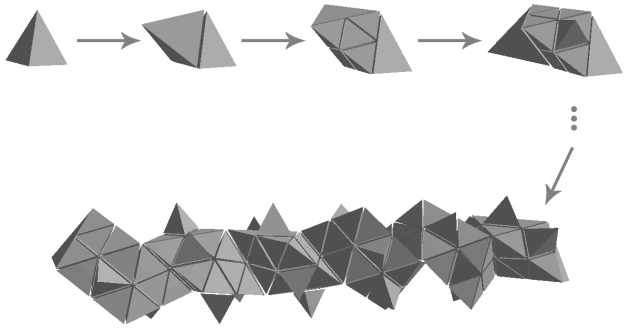

Figure 3: The growth of a tetrahedral mesh by iteratively applying the rules from a face encoding ruleset like the one shown in Table 1

These grammars can be evolved by treating the rulesets as genotypes, and the tetrahedral mesh which results after a fixed number of iterations as the phenotype. Rulesets can be encoded as simple linear strings of characters subject to mutation and crossover. Mutation on a grammar genotype affects only the right hand side of a rule, and can either change the rule (i.e. $grow(A, B, D) \rightarrow relabel(A)$, with extra labels added or removed as necessary) or change a label in the rule (i.e. $grow(A, B, D) \rightarrow grow(E, B, D)$. Single point crossover grabs a subset of production rules from one parent, and the remainder from a second. An evolving population simply consists of collection of grammar genotypes.

4. GROWING AND EVOLVING SOFT ROBOTS

Using the encoding described above as a genetic basis we can perform evolution on the resulting soft body phenotypes, evaluating them the PhysX physics engine. For this research our interest is in locomotion, and the ability of a soft body phenotype to locomote was measured by periodically varying the stiffness of the tetrahedral mesh, measuring the absolute displacement of the body after a fixed interval. Details of our approach are described in [23]. Hiller and Lipson used a similar approach in their amorphous robots [12], which they were able to physically implement using closed cell foam and a vacuum chamber [13].

4.1 Experimental Details

For all the experiments described below, we used a fixed population size of 20 and 50% elitism. Parents were cho-

sen with a simple fitness proportional selection. 40% of offspring were produced via crossover, and the remainder via mutation. An edit-distance diversity metric was employed to prevent multiple neutral mutations of a single genotype schema to proliferate.

Each individual was evaluated by applying its genotype grammar rewrite rules a fixed number of times (either 20 or 40) in order to produce a phenotype, and then placing the resulting robot in the PhysX environment. Tetrahedral mesh stiffness was then cycled between 0.99 and 0.80, with a period of 200 time steps, and the displacement vector over each cycle recorded. Total distance traveled during a 4000 time step window was then computed by adding the displacement vectors.

In order to prevent the common but pathological trait of toppling (first made famous by Sims' evolved agents [26]), the displacement vectors were passed through a low-pass filter which penalized short term high velocities associated with toppling, while preserving longer term slower velocities.

5. STATIC DEVELOPMENTAL TIMING

As a demonstration of the pathologies of ontogenic trajectories, we first ran a a suite of experiments with hard-coded ontogenic endpoints. In these experiments, only one phenotypic fitness along the trajectory, the last, was measured and used for evolutionary feedback. Multiple experiments were run at 20, 40, 100, and 200 rule applications of the grammar. All experiments ran for 500 generations.

Recall that despite the single evolutionary measure of fitness there is of course an underlying ontogenic process, meaning that every intermediate phenotype is itself a complete and possibly viable body shape. In a *post hoc* fashion, we can therefore measure the fitness of the entire *ontogenic trajectory* for a given genotype. Since these intermediate fitnesses are never selected for, there is no expectation that any other point along the trajectory should have any significant fitness.

The graphs in Figure 4 illustrate the ontogenic fitness trajectories of several representative "best" solutions from our experimental suite. The trajectory graphs were generated *after* evolution by placing the grammar genotype into our physics engine and evaluating the fitness of the phenotype resulting from every single incremental grammar iteration. Fitness trajectories were deliberately measured beyond the static ontogenic endpoint used for evolution (i.e. the phenotypes were "overgrown"). The x axis values are normalized around the "endpoint fitness" – that is the fitness used for evolutionary feedback.

One property common to all the trajectories, in support of the fact that non-terminal ontogenic fitnesses aren't selected for, is the fact that fitness does not monotonically increase over the course of the ontogenic trajectory. However, rather than always being highest at the ontogenic endpoint, as in the top image, they can instead exhibit high variation. Sometimes, as in the middle image, there is a sharp jump in fitness near the iteration limit, and fitness remains quite stable for several additional iterations past the static ontogenic endpoint before gradually decaying. In other cases (bottom), fitness builds gradually (and non-monotonically) up to the endpoint, and then drops to around 30% of nominal fitness for the next 10 iterations, and actually jumps up above 50% of nominal fitness after 15 iterations. In a third case (lower figure), there are ontogenic stages before and after the iteration limit which are twice as high as the evolved fitness value.

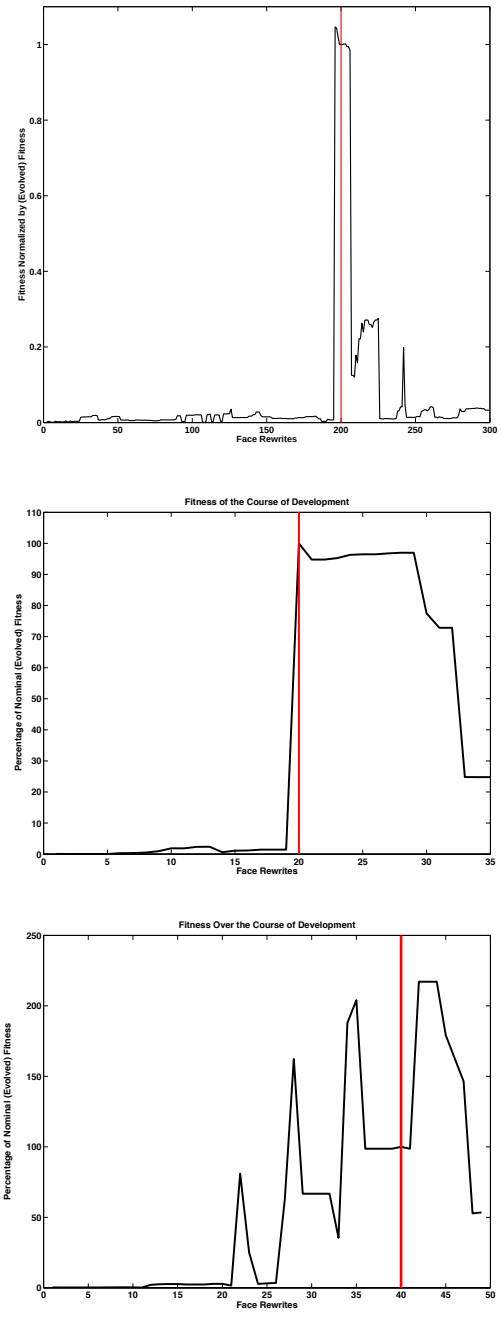

Figure 4: Ontogenic fitness trajectories created by measuring intermediate phenotypic fitness over the course of development, and then normalized by the fitness measured at the fixed iteration limit (vertical red line). Although during evolution fitness is only measured at that iteration limit, fitness sometimes remains stable as development proceeds past the limit (second graph), and fitness is sometimes higher at earlier and later developmental stages.

This illustrates that even though evolution is selecting for fitness at a specific ontogenic endpoint, this static developmental limit is not a "magic number", and in fact sometimes misses the fitness peak of the ontogenic trajectory by a wide margin. These results support and build upon Viswanathan's claims about the potential deficiencies of developmental approaches [29].

It is worth noting that all generative encodings have a modest intrinsic ability for heterochrony, for instance by incorporating several "no-op" rules into development (for instance $A \rightarrow relabel(A)$. However, this type of internal heterchronic mechanism cannot easily be directly controlled, and can only be accomplished by in essence making the developmental process less efficient.

6. INTRA-EVOLUTIONARY SCALING OF ONTOGENIC ENDPOINTS

Rather than providing a pessimistic outcome however, these arguments strongly suggest that a more adaptive approach which is more sensitive to fitness over the course of development might help improve developmental encodings. The basic premise of our scaled approach is to gradually increase the developmental duration of the evolved phenotypes by slowly incrementing the number of grammar rewrites over the course of evolution. This is in essence a middle ground between static ontogenic endpoints and Visvanathan's exhaustive and computationally intensive "informed Genotype-Phenotype map".

The details of the algorithm are as follows. At the outset, the ontogenic endpoint (i.e. the number of grammar expansions before fitness evaluation), $curEndPt$, is set to some initial value $InitialEndPt$. All genotypes in the population are then expanded $curEndPt$ times before being evaluated for fitness.

Evolution proceeds apace for $ExpnInterval$ generations, after which the ontogenic endpoint $curEndPt$ is increased by $ExpnRate$. This scaling of ontogenic endpoints is repeated every $ExpnInterval$ generations.

Bear in mind, as illustrated by the ontogenic trajectories in Figure 4, there is a risk that increasing the developmental limit of a particular grammar may in fact decrease its fitness. Allowing $curEndPt$ to increase at this fixed interval runs the risk that the fitness of the evolving population may in fact *decrease* over time. We therefore add a "ratchet" which only allows $curEndPt$ to increase once the current best fitness of the population has surpassed the former (pre-increase) best fitness, $overallBestFitness$.

Algorithm 1 describes the process more formally. Note that when $InitialEndPt$ is set to a static value, $ExpRate$ is set to 0 and $ExpnInterval > MaxGens$, the algorithm reverts to the static developmental timing of Section 5

Figure 5 compares the ontogenic trajectory fitness (top) vs the current ontogenic endpoint ($curEndPt$) over the course of a single evolutionary run, illustrating how intra-evolutionary increases in endpoints time cause a corresponding decrease in overall fitness. At these points the "ratchet" kicks in, suppressing further increases in $curEndPt$ until the overall fitness has increased beyond pre-increase levels.

One alternative to this approach would be to directly encode developmental endpoint into the grammar genome itself, as an evolvable parameter not unlike Hornby's parametric L-Systems. As we will see below, there are often

Algorithm 1 Evolutionary Scaling of Grammar Expansions

```
curEndPt ← InitialEndPt
curGen ← 0
overallBestFitness ← 0
while curGen < MaxGens do
  for each genotype g in Population P do
    curPhenotype ← Grow(g, curEndPt)
    Evaluate(curPhenotype)
  end for
  curGen ← curGen + 1
  curInterval ← curInterval + 1
  P ← MakeNewPop(P)
  if (MaxFitness(P) > overallBestFitness)
  && (curInterval > ExpnInterval) then
    curEndPt ← curEndPt + ExpnRate
    overallBestFitness ← MaxFitness(P)
    curInterval ← 0
  end if
end while
```

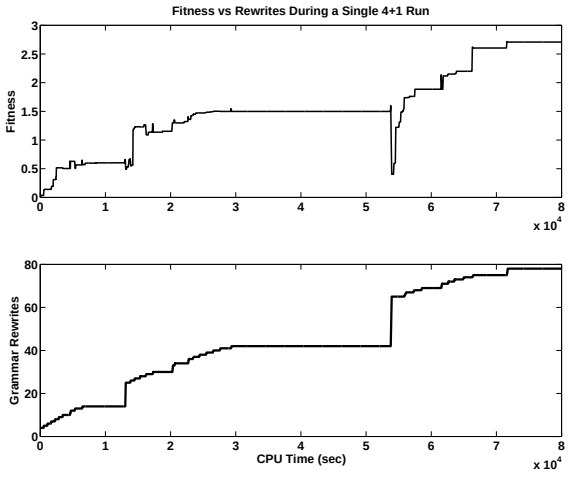

Figure 5: An illustration to the intra-evolutionary scaling of ontogenic endpoints. The maximum number of grammar rewrites (lower figure) starts out quite low, and then is incrementally raised every 20 generations. Often, increases in the developmental duration cause a drop in phenotypic fitness, as the larger phenotypes are no longer as efficient at motion. Evolution proceeds with this new developmental duration until fitness exceeds the prior maximum, at which point the developmental duration is increase again.

Name	InitialEndPt	ExpRate	ExpnInterval
(Static 200)	200	n/a	n/a
(Static 100)	100	n/a	n/a
(20+5/100)	20	5	100
(4+1/20)	4	1	20
(4+2/20)	4	2	20

Table 2: Test suite for comparing static and scaled ontogenic endpoint schemes

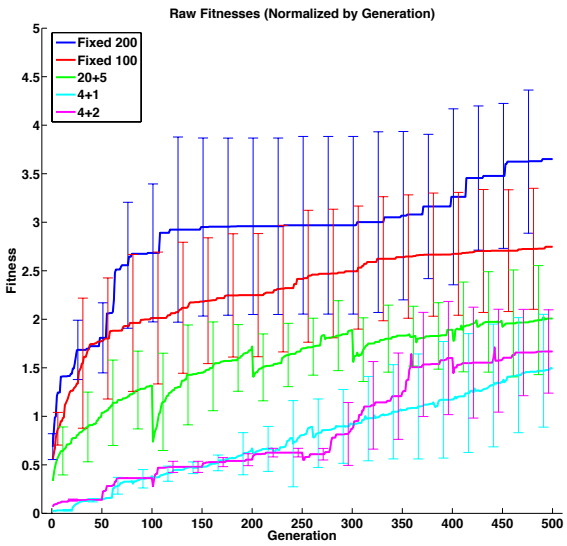

Figure 6: A comparison of static vs scaled ontogenic endpoints, as described in Table 2. When using elapsed generations as an x axis, it appears as if static endpoints are clearly superior. As we discuss in the text, however, this in not a fair comparison.

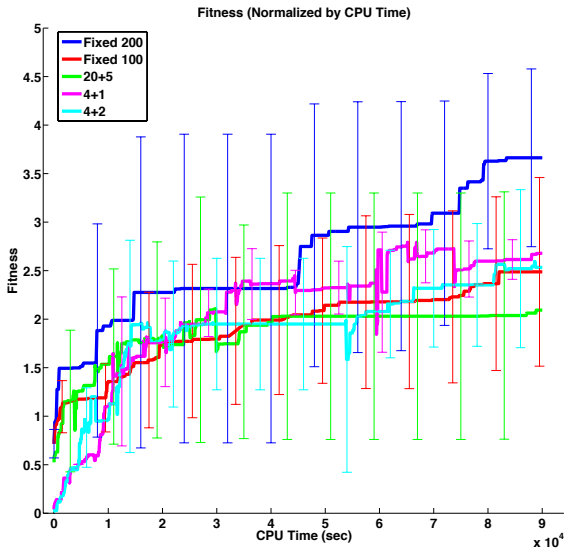

Figure 7: The same data from the test suite as in Figure 6, now scaled by elapsed CPU "wall time". In the early stages of evolution the scaled endpoint approach produces considerably smaller tetrahedral meshes, which are significantly faster to simulate. They can therefore iterate through the early generations much more quickly than the static endpoint approach.

short-term gains associated with long initial developmental periods, and our concern is solutions in such an approach would exploit long developmental durations at the expense of longer term progress.

6.1 Scaled Development Results and Analysis

We ran three sets of scaled developmental timing experiments, and compared them against two of the static timing setups described in Section 5, as summarized by Table 2.

6.1.1 *Effects on Computational Efficiency*

When comparing the static and scaled developmental timing schemes on a per-generation basis, as in Figure 6, it would appear that the static scheme has a clear advantage. However, this analysis is slightly misleading, in the sense that this is not a purely objective comparison. Tetrahedral meshes generated with 20 iterations generally have fewer tetrahedra than those generated with 40, or 100 iterations. Since computational complexity is proportional to the the number of tetrahedra, a comparison of the schemes vs elapsed CPU "wall time" is a more fair means of comparison.

Figure 7 presents the same data as in Figure 6 using CPU time instead of generations as a time scale. Doing so illustrates how the scaled developmental schemes compare more favorably against the static approach, particularly the static-100 approach. Because of the lower computational cost of smaller meshes, the scaled endpoints scheme iterates through considerably more generations than the static endpoint schemes, particularly during the early stages of evolution.

One explanation for the continued strength of the static-200 scheme might be the choice of fitness function used in the experiment. Distance traveled by the evolved phenotype is not normalized by body size (along any dimension), and so there is a built-in bias toward larger structures.

6.1.2 *Effects on Complexity of Solutions*

The stark difference in the evolved mesh complexity between the two schemes is best illustrated by Figure 9, which compares the number of tetrahedra in the phenotype meshes between schemes. The mesh complexity of the static scheme is initially quite high, and then decreases over the course of evolution. The scaled endpoint approach, by contrast, gradually (but non-monotonically) increases mesh complexity. This suggests that the static approach initially creates overly large meshes, and then gradually finds smaller, more fit meshes. The scaled approach by contrast starts with initially quite simple meshes (only a handful of tetrahedra) and slowly ramps up mesh complexity as the ontogenic endpoint scales upwards.

6.1.3 *Effects on Parsimony of Solutions*

The value of the scaled developmental duration approach is reinforced when comparing the total number of rewrites required to produce the best fit individual of each genera-

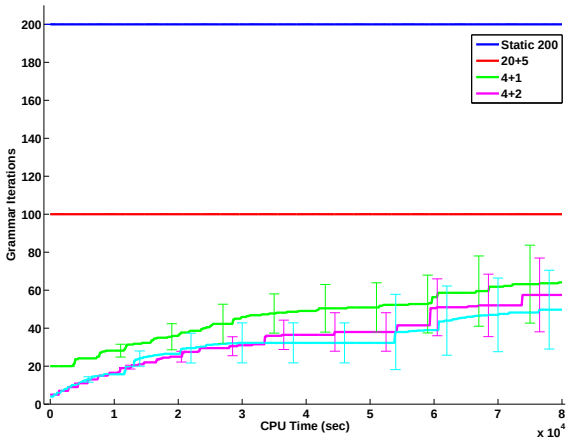

Figure 8: Unlike the static methods, in our scaled approach the number of grammar rewrites slowly increases over time, and achieves competitively fit results with significantly fewer rewrites.

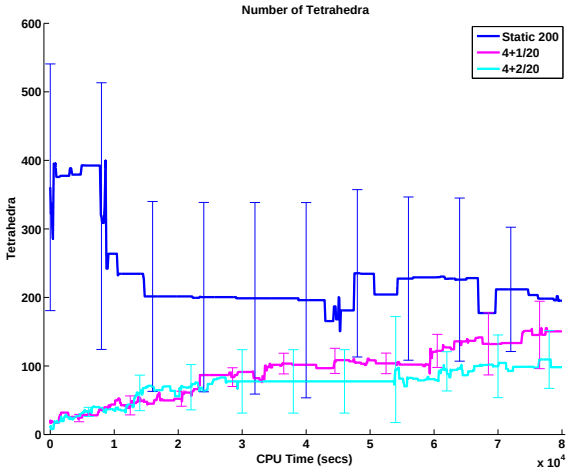

Figure 9: Comparison of tetrahedral mesh complexity (number of tetrahedra) between static and scaled developmental endpoints. The mesh complexity of the static approach generally decreases over the course of evolution whereas the complexity of the scaled approaches increase.

tion, as illustrated by Figure 8. By their nature, the static approaches have a constant number of rewrites over the entire course of evolution. The scaled approaches by comparison begin with very low developmental durations, and slowly increase the number of rewrites over the course of evolution. As is apparent, by the end of evolution the scaled results are producing comparably fit results while requiring considerably fewer rewrites. This underscores that the scaled approach solutions are in a sense more *parsimonious* than the static solutions.

7. CONCLUDING REMARKS

This paper addressed the inherent complexities and pathologies associated with ontogenic timing in generative developmental encodings – the so-called "halting problem" in GDS. Our claim is that encodings which fix the duration of development *a priori* are not leveraging the full potential of development, and can often produce results which are non-optimal. We have described a system which slowly increases the duration of development over the course of evolution. Our aim is not to demonstrate that our approach is superior to other developmental approaches, but rather to use this model to hold a light to some less explored aspects of the role of developmental timing in developmental systems.

The results of our experiments do just this. On one hand, an analysis of our technique based upon overall fitness produces equivocal results: even when using CPU "wall" time as a time scale, our scaled approach merely matches the performance of a static 100-rewrite evaluation, and slightly underperforms compared to a 200-rewrite scheme. The benefits of our scaled approach are instead highlighted when looking in more detail at the underlying processes and products. Our approach produces results which are more parsimonious (require fewer rewrites) and less "overbuilt" (fewer tetrahedra) than those produced by static approaches. This adds several new dimensions to the field's understanding of the value of adaptive developmental timings.

More broadly, this paper ties in to larger ideas of Evolutionary Development (Evo-Devo) as well. By gradually increasing the duration of development, we are in a sense building in a mechanism for *recapitulation*. Successful "fully grown" phenotypes are emerge under earlier, shorter developmental trajectory periods become the intermediate *ontogenic* building blocks within longer developmental trajectories later on in evolution. In a sense, therefore early phenotypes are encapsulated into the ontogenies of later phenotypes – in other words ontogeny recapilutates phylogeny. While recapitulation in its strictest Haeckelian sense has been discredited, the notion that an organisms morphology is strongly determined by its order within a phylogeny maintains some value in modern Evo-Devo.

This research also raises several new questions which we hope to explore in the future. Foremost among them is to perform a more careful comparitive analysis of the shape of ontogenic fitness trajectories between fixed and scaled developmental duration schemes. Our suspicion is that the scaled approach leads to more stable trajectories than the static approach.

8. REFERENCES

[1] J. Auerbach and J. Bongard. Evolving complete robots with cppn-neat: the utility of recurrent

connections. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1475–1482. ACM, 2011.

[2] J. E. Auerbach and J. C. Bongard. Evolving cppns to grow three-dimensional physical structures. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 627–634, New York, NY, USA, 2010. ACM.

[3] J. Bongard. Evolving modular genetic regulatory networks. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 2, pages 1872 –1877, 2002.

[4] J. Bongard and R. Pfeifer. *Morpho-functional Machines: The New Species (Designing Embodied Intelligence)*, chapter Evolving complete agents using artificial ontogeny, pages 237–258. Springer-Verlag, Berlin, 2003.

[5] J. C. Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the 2002 IEEE Conference on Evolutionary Computation (CEC2002)*, pages 1872–1877, Piscataway, NJ, 2002. IEEE Press.

[6] J. C. Bongard and R. Pfeifer. Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In L. Spector et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 829–836, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[7] A. Chavoya and Y. Duthen. Using a genetic algorithm to evolve cellular automata for 2d/3d computational development. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 231–232. ACM, 2006.

[8] A. Devert, N. Bredeche, and M. Schoenauer. Robustness and the halting problem for multicellular artificial ontogeny. *Evolutionary Computation, IEEE Transactions on*, 15(3):387 –404, june 2011.

[9] D. Federici and K. Downing. Evolution and development of a multicellular organism: scalability, resilience, and neutral complexification. *Artificial Life*, 12(3):381–409, 2006.

[10] S. Gould. *Ontogeny and Phylogeny*. Belknap Press, 1985.

[11] M. Hemberg and U.-M. O'Reilly. Extending grammatical evolution to evolve digital surfaces with genr8. In *EuroGP*, 2004.

[12] J. D. Hiller and H. Lipson. Evolving Amorphous Robots. In H. Fellermann, M. Dörr, M. M. Hanczyc, L. L. Laursen, S. Maurer, D. Merkle, P.-A. Monnard, K. Stoy, and S. Rasmussen, editors, *Artificial Life XII: Proceedings of the Twelfth International Conference on the Simulation and Synthesis of Living Systems*, pages 717–724. MIT Press, Cambridge, MA, Aug. 2010.

[13] J. D. Hiller and H. Lipson. Morphological evolution of freeform robots. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 151–152, New York, NY, USA, 2010. ACM.

[14] T.-H. Hoang, R. McKay, D. Essam, and X. Nguyen. Learning general solutions through multiple evaluations during development. In G. Hornby, L. Sekanina, and P. Haddow, editors, *Evolvable Systems: From Biology to Hardware*, volume 5216 of *Lecture Notes in Computer Science*, pages 201–212. Springer Berlin Heidelberg, 2008.

[15] G. S. Hornby. Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO-2005)*, pages 1729–1736, New York, NY, USA, 2005. ACM Press.

[16] G. S. Hornby and J. B. Pollack. The advantages of generative grammatical encodings for physical design. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 600–607, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 2001. IEEE Press.

[17] G. S. Hornby and J. B. Pollack. Evolving l-system to generate virtual creatures. *Computers and Graphics*, 25(6):1041–1048, 2001.

[18] J. D. Lohn, G. S. Hornby, and D. S. Linden. An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission. In U.-M. O'Reilly, R. L. Riolo, T. Yu, and B. Worzel, editors, *Genetic Programming Theory and Practice II*. Kluwer, 2005.

[19] J. B. Pollack, H. Lipson, G. Hornby, and P. Funes. Three generations of automatically designed robots. *Artifial Life*, 7(3):215–223, 2001.

[20] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer–Verlag, New York, USA, 1990.

[21] R. A. Raff. *The Shape of Life: Genes, Development, and the Evolution of Animal Form*. University Of Chicago Press, June 1996.

[22] M. K. Richardson. Heterochrony and the phylotypic period. *Developmental Biology*, 172:412–421, 1995.

[23] J. Rieffel, D. Knox, S. Smith, and B. Trimmer. Growing and evolving soft robots. *Artificial Life*, to appear, 2013.

[24] J. Rieffel, F. Valero-Cuevas, and H. Lipson. Automated discovery and optimization of large irregular tensegrity structures. *Computers & Structures*, 87(5-6):368 – 379, 2009.

[25] K. Sims. Evolving 3d morphology and behavior by competition. In R. Brooks and P. Maes, editors, *Artificial Life IV Proceedings*, pages 28–39. MIT Press, 1994.

[26] K. Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM Press, 1994.

[27] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life*, 15:185–212, April 2009.

[28] T. Steiner, Y. Jin, and B. Sendhoff. Evolving heterochrony for cellular differentiation using vector field embryogeny. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 571–578. ACM, 2010.

[29] S. Viswanathan and J. Pollack. How artificial ontogenies can retard evolution. In *Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 273–280. ACM, 2005.