

# The Emergence of Ontogenic Scaffolding in a Stochastic Development Environment

John Rieffel<sup>1</sup> and Jordan Pollack<sup>1</sup>

DEMO Lab, Brandeis University, Waltham MA, 02454, USA  
{jrieffel,pollack}@cs.brandeis.edu  
<http://demo.cs.brandeis.edu>

**Abstract.** Evolutionary designs based upon Artificial Ontogenies are beginning to cross from virtual to real environments. In such systems the evolved genotype is an indirect, procedural representation of the final structure. To date, most Artificial Ontogenies have relied upon an error-free development process to generate their phenotypic structure. In this paper we explore the effects and consequences of developmental error on Artificial Ontogenies. In a simple evolutionary design task, and using an indirect procedural representation that lacks the ability to test intermediate results of development, we demonstrate the emergence of ontogenic mechanisms which are able to cope with developmental error.

## 1 Introduction

Recently, evolved designs have begun to cross the boundary from the virtual to the real [1, 2]. Many of these designs are based upon Artificial Ontogenies [3, 4], which use an *indirect encoding* of the evolved object. Between genotype and phenotype lies some developmental process responsible for assembling the phenotypic structure by interpreting instructions contained in the genotype.

While many such systems take noisy physics into account when evaluating the fully developed phenotype [5–7], the problem of noise during development is yet to be addressed, and to date, Artificial Ontogenies have not been shown to be adaptive to errors caused by noisy development environments. With the real-world assembly of evolved designs in mind, our interest here is on the ability of Artificial Ontogenies to adapt to error during development. This is a line of inquiry intimated by Stanley and Miikkulainen in their recent survey [4].

As we show, developmental error can complicate an otherwise trivial design task. Error during development results in a stochastic process wherein each genotype, instead of reliably developing into a single phenotype, develops into an entire *distribution* of heterogeneous phenotypic structures, with a corresponding range of fitness values. As such, a credit-assignment problem arises: when a genotype develops into a variety of heterogeneous phenotypes, how should the entire range of related fitnesses be attributed to that genotype?

In this paper we begin to explore whether, without incorporating tests into the developmental system, there is enough information available to the evolutionary process to allow for mechanisms to emerge which can cope with stochastic

development. We first evolve an indirect encoding in an error-free development environment and demonstrate its failure when assembled in a stochastic environment. We then incorporate noise into the development environment used *within* the evolutionary process. In this setup we are able to observe the emergence of ontogenic mechanisms capable of overcoming developmental error.

## 2 Theory and Background

Artificial Embryogenies [4] distinguish themselves from other forms of evolutionary computation by treating the genotype as an *indirect*, or *procedural* encoding of the phenotype. The genotype is decoded and transformed into a phenotype by means of some developmental process. As a result, a single-point change to the genotype can have multiple (or zero) effects upon the phenotype. This abstraction layer between genotype and phenotype allows for quite a bit of flexibility during evolution, and has several demonstrated advantages [8, 9, 4, 3, 10]. An advantage of indirect encodings that we are particularly interested in is their ability to specify intermediate morphological elements that are useful for ontogenesis, but that do not exist in the final phenotype.

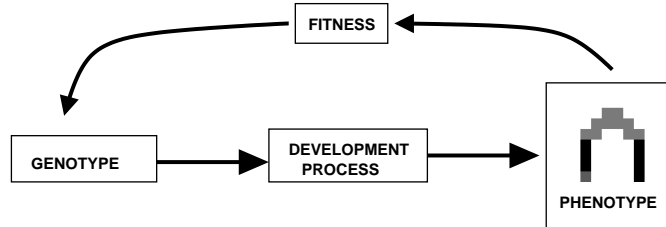
### 2.1 Genotypes as Assembly Plans

In distinguishing the direct encodings used in traditional GAs from the *indirect* encodings used by Artificial Ontogenies, it is informative to consider the distinction between a blueprint and an assembly plan. A direct encoding is a *descriptive* representation. It is like a blueprint in the sense that it conveys what the phenotype should look like, but carries no information about how to build it (or whether in fact it can be built at all.) Examples of evolved direct encodings include Lipson's Golems [2] and Fune's LEGO structures [11]. Indirect encodings, on the other hand, provide no information about what the final structure should look like. Rather they are a *procedural* representation, and like an assembly plan, give specific instructions on how to build the structure step by step.

When their genomes are described as assembly plans, Artificial Ontogenies can be considered a form of Genetic Programming (GP) [12]. The genome, either linear or in the form of the tree, consists of loci which are instructions to some ontogenic agent. This agent (which is not necessarily external to the developing structure), interprets each instruction and builds the emerging structure from raw materials accordingly. In the case of Hornby [8, 7, 2], the instructions are commands to a LOGO-like turtle which builds three dimensional structures out of voxels. In the case of Toussaint [9], the instructions are for a system which draws three-dimensional plants from component stems and leaves. Assembly plans can be categorized as either *ballistic* or *adaptive*. Ballistic assembly plans have no internal feedback mechanisms - they proceed uninterrupted until done, regardless of the results of each action. Adaptive assembly plans, on the other hand, are able to measure the results of their executed instructions, and change their behavior accordingly.

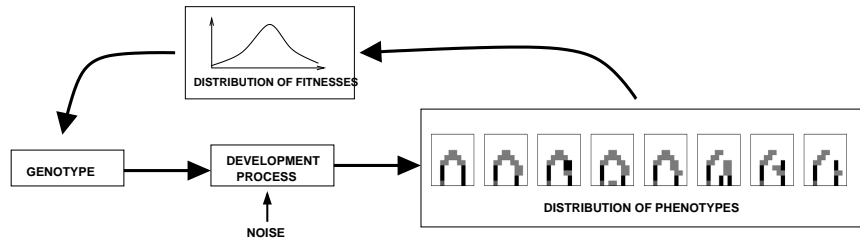
## 2.2 The Effects of Noise during Development

Most Artificial Embryogenies (Fig. 1) rely upon a deterministic development process. As such, there is a one-to-one relation between genotype and phenotype: a given genotype will always develop into the same phenotype.



**Fig. 1.** In a simple Artificial Ontogeny with a deterministic development, each genotype consistently develops into a single phenotype and associated fitness

Introducing error into development causes a one-to-many genotype/phenotype relationship. Since the result of each stage of the ontogeny is predicated upon the result of the previous stage, an early error can drastically affect the outcome of the final phenotype. Under these conditions a genotype may produce any number of phenotypically heterogeneous results, as illustrated by Fig. 2. [13] provides a more nuanced treatment of this phenomenon.



**Fig. 2.** In a Artificial Ontogeny with a noisy development environment, each genotype can develop into an entire range of phenotype, with a corresponding range of fitnesses

One possibility for overcoming developmental error is to include some form of test into the genotype's set of primitive instructions. However, incorporating tests into each step of an ontogeny can be time consuming, particularly in the context of an evolutionary search spanning thousands or millions of generations. Another way to handle stochastic ontogenies might be to use systems capable of modularity and parallelism such as generative grammars [8] or genetic regulatory networks [10]. Like tests, however, such methods come at the expense of simplicity of the ontogenic process.

Before exploring more complex, albeit powerful, genotypes and ontogenies it is worthwhile to first explore the capabilities and limits of a simple linear, ballistic assembly plan, whose only feedback exists at the evolutionary scale.

### 2.3 Measuring Fitness Distributions

Rather than give each genotype only one chance to stochastically develop into a phenotype, it may be more informative to allow each genotype multiple stochastic developments. A genotype will then produce an entire *distribution* of phenotypes, with a corresponding range of fitness values, per Fig. 3. Statistical measurements of the resulting distribution can then be used to measure the fitness of the genotype.



**Fig. 3.** A noisy development environment leads to a distribution of phenotypic fitnesses. Yield is the frequency with which the distribution reaches the maximum fitness

In the case where there is an achievable maximum fitness, gradient can be further induced by considering *yield*: the frequency with which the maximum fitness is attained (see Fig. 3). To illustrate this, consider the case where there is a particular evolutionary goal in mind, such as the pre-defined letter shapes on a grid in Kumar and Bentley’s recent work [3]. In this context, yield can be described as the percentage of times that a given assembly plan is able to successfully generate the goal phenotype.

With such a range of different statistical measurements available to compare genotypes, choosing a specific scalar fitness function which somehow weighs and combines the measurements into a single informative value can be difficult. In this situation, Evolutionary Multi-Objective Optimization (EMOO) [14, 15] can prove useful.

EMOO allows each measurement to exist as an independent objective. Instead of a scalar fitness value, each genotype is given a set of fitness values, one for each objective. When comparing two sets of objective values, one is said to *Pareto dominate* the other when it is at least as good in all objective dimensions, and better in at least one dimension. Given a population of individuals, the *Pareto front* consists of the set of individuals that are not dominated by any other individuals. A more detailed mathematical explanation of EMOO can be found in [14] and [15].

### 3 Experiments

The goals of our experiments are twofold: first to demonstrate that “naive” indirect encodings evolved in an error-free development environment are brittle in the face of error during ontogeny; and secondly to show how indirect encodings evolved *within* a stochastic environment are able to adapt to error, and reliably produce fit phenotypes.

We phrase our problem as a type of Genetic Programming [12] in which we are evolving a linear assembly plan to build a predefined “goal” structure. In this case, we chose an arch (Fig. 4), in part for the expected level of difficulty, and in part for historical reasons - its presence in Winston’s seminal work on Machine Learning [16].



**Fig. 4.** The goal structure. Note: vertical bricks are black, horizontal bricks are grey

The genotype consists of a linear set of parameterized instructions for a LOGO-turtle like builder, the ontogenic agent. The turtle is capable of moving along a vertical 2-D plane, and placing and removing  $2 \times 1$  bricks within the plane. Table 1 lists the instructions used. Note that assembly plans are completely ballistic: there are no instructions that can test the state of the world or the results of the most recent instruction.

**Table 1.** Parameterized Assembly Instructions

Instruction	Parameters
(M)ove	+2, +1, -1, -2
(R)otate	+90, -90, +180
(P)ut Brick	(a)head, to (r)ight, to (l)eft, (b)ehind
(T)ake Brick	(none)

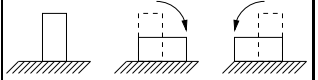
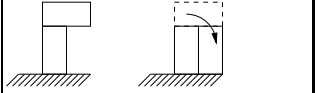
Because genotypes are linear sequences of instructions, they are amenable to both crossover and mutation. In order to allow for a broader syntactic range of acceptable genotypes, the builder is tolerant of redundant instructions (such as putting a brick where a brick already exists), as well as instructions which would force it off of the grid.

### 3.1 Physics

Bricks placed by the turtle are subject to a simple physics model. They must either be supported by the floor of the plane or by another brick. Bricks unsupported from below will fall until they hit a supporting surface.

By adding noise to the physics of the development environment, we can induce developmental errors. Bricks placed vertically on a surface have a 50% chance of staying in place, and a 50% chance of falling to either side. Similarly, bricks placed horizontally such that they are cantilevered have a 50% chance of remaining in place and a 50% chance of falling. Naturally, surrounding bricks may act as supports, and reduce the chance that a brick will fall. Bricks that fall will drop until they find a resting place. Once a brick has settled it is considered “glued” in place until it is removed or one of its supporting bricks is removed. Table 2 summarizes the rules of the stochastic physics. Note that the turtle itself is imperturbable. Its position on the plane remains constant regardless of whether the brick it has placed falls or not.

**Table 2.** Basic Rules for Stochastic Physics

Vertical Bricks have a 50% chance of falling to either side.	
Cantilevered Bricks have a 50% chance of falling	

The developmental error of our assembly is therefore of a very specific nature: each instruction in the assembly plan is always reliably executed by the builder, but the *result* of that instruction may vary.

### 3.2 Algorithm

As mentioned above, we chose to phrase the problem as one of Evolutionary Multi-Objective Optimization (EMOO) [14, 15]. The specific objectives vary between experimental setups, and are discussed in detail for each.

**Evaluation** Individuals are evaluated by interpreting their assembly plans within the specified environment and measuring the properties of the resulting structure. For non-stochastic environments, each assembly plan only needs to be build once. For stochastic environments, assembly plans are built several times in order to gather statistical properties of their phenotypic distribution.

**Generation and Selection** Population size is variable - new children are added and evaluated until the population is doubled. New individuals are generated by a combination of two-point crossover (70%) and single-point mutation (30%). Once the new population has been generated and evaluated, the population is culled by and keeping only non-dominated individuals, i.e. the Pareto front.

### 3.3 Evolving Without Developmental Noise

As a first demonstration, consider a “naive” assembly plan evolved in an error-free development environment. The objectives used for this run are as follows:

- length of genome (shorter is better)
- genotypic diversity
- number of squares missing from goal structure (fewer is better, 0 is best)
- sum of number of missing squares and extra squares (fewer is better, 0 is best)

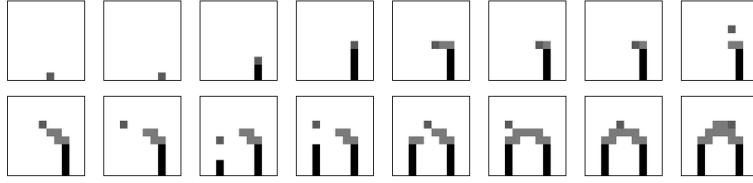
The length objective exists in order to find minimal solutions, as well as a deterrent to bloat [17, 18]. Because of the small number of objectives, and due to the propensity of the system to find a large number of genotypically similar, and therefore redundant solutions, we follow the lead of [17] by adding a diversity metric. This metric is calculated as the average hamming distance between the genome and all other genotypes in the population.

Treating the goal and result structures as 2-D bitmaps, the third objective can be calculated as the sum of the bitwise AND of the goal and the inverse of the result,  $\sum_{i,j} (goal(i,j) \otimes \neg result(i,j))$ , and the fourth objective as the sum of the bitwise XOR of the goal structure and the result:  $\sum_{i,j} (goal(i,j) \oplus result(i,j))$ . As an example, consider the leftmost structure in Fig. 6: three squares are absent from the goal structure, and there are nine extra squares. The third objective would therefore be 3, and the fourth objective would be 12.

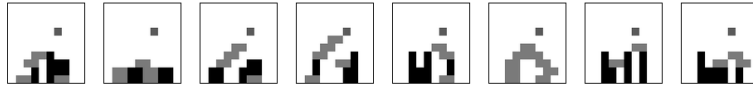
The last metric, which adds the number of missing square and extra squares, may seem cumbersome, but earlier attempts which simply tried to minimize the number extraneous bricks ended up rewarding long, diverse assembly plans which simply moved about but did not place any bricks. By combining missing squares and extra squares, this behavior is avoided.

**Results** With this set-up, the system is able to find a minimal assembly plan capable of building the arch in Fig. 4, as shown in the sequence of frames in Fig. 5. The corresponding genotype is: [R(+90) M(-2) P(r) P(a) P(l) M(-1) R(+90) M(-2) P(r) M(+2) P(l) P(b) P(r) P(b) M(-2) P(b)]

Not surprisingly, when that same minimal assembly plan is then built with a noisy development environment it completely fails to build the goal structure - even given repeated attempts. Figure 6 shows a sample of the resulting phenotypes.



**Fig. 5.** “Naive” Assembly Plan for Arch. Frames are read left-to-right, top to bottom. The dark grey square is location of the builder



**Fig. 6.** A sample of the resulting phenotypes when built with noise

### 3.4 Evolving With Developmental Noise

Our second approach is to integrate a noisy development environment into the evolutionary process itself - such that every candidate genotype is evaluated in the noisy physics. Instead of being built once, each assembly plan is evaluated 50 times, and statistical measures used as evolutionary objectives. The set of measurements that most consistently yielded the best results are:




- length (shorter is better)
- number of missing squares:
  - best, average and yield percentage (no missing bricks)
- sum of extra squares and missing squares:
  - best, average and yield percentage (perfect structure)

Note the absence of the diversity metric used in the first experiment. Such a large number of objectives here results in a relatively large Pareto front with a sufficient amount of diversity.

**Results** The evolutionary system described above is typically able to generate assembly plans with yields above 70%. The result we present below is 82 instructions long, and reached a 70% yield during its 50 evolutionary evaluations. When evaluated a further 500 times, its yield drops to 65%. This discrepancy can be attributed to the relatively small sample size used in evolution. Table 3 below shows some samples of the range of phenotypes produced by this assembly plan over the course of multiple developments. It is able to perfectly build the goal structure (far left) 65% of the time. It was able to produce a structure without any squares missing from the structure (middle figures) an additional 8% of the time. The remainder of results (right hand figures) contained some, but not all, of the goal structure.



**Table 3.** Samples of the distribution of phenotypes of the robust assembly plan

		
Structure Intact: 73%		Partial Structure: 27%
Perfect:65%	Extra Bricks:8%	

In a typical run, by the time a genotype with 64% yield is achieved, the evolution has run through 26300 generations, and more than 100,000,000 genotype evaluations (where each genotype is evaluated 50 times!), and the population consists of more than 3000 individuals. Beyond this point we therefore suspect that the limitation on further maximizing yield lies largely in the computational effort involved in evaluating such large populations.

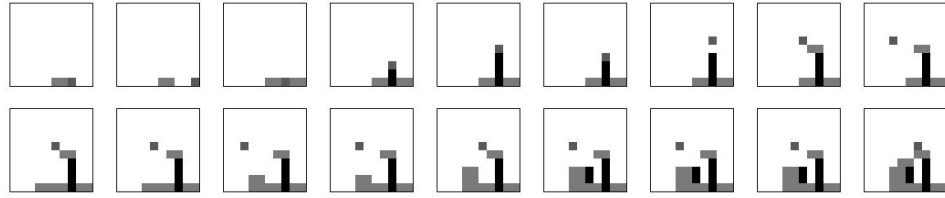
**Emergence of Ontogenic Scaffolding** When a genome’s fitness is based upon the statistical properties of its phenotypic distribution we can think of the role of evolution as learning to shift phenotypic fitness distributions, rather than individual values, towards the optimal. For instance, given two genotypes, the one that on average produces more fit individuals can be considered the better one. In this context, the value of the indirect encoding as assembly plan comes into play. Because assembly plans have the ability to describe *how* a structure is to be built, they can include instructions which place intermediate elements into the structure whose role is to ensure that later elements of the structure stay in place. We call these elements *ontogenic scaffolding*. Once all of the elements of the final structure have been placed, the ontogenic scaffolding can be removed, leaving behind a stable final structure. This ontogenic scaffolding is evident in the results above.

Consider the frames in Figs. 7 through 9 below, which show a typical development from the robust assembly plan discussed above. (Animated versions of these images can be found at <http://www.cs.brandeis.edu/~jrieffel/arches.html> )

The assembly begins with Fig. 7. The assembly plan first places horizontal bricks to the left and right of what will become the first leg of the structure. Their presence guarantees that the leg will stay in place. The plan then places the first and second vertical bricks - both parts of the goal structure. Note the “redundant” instruction in the sixth frame for Fig. 7. Although it appears extraneous in this particular sequence, it proves useful in situations where the first attempt at laying the second brick fails: in which case the fallen brick ends up acting as scaffolding for the subsequent attempt.

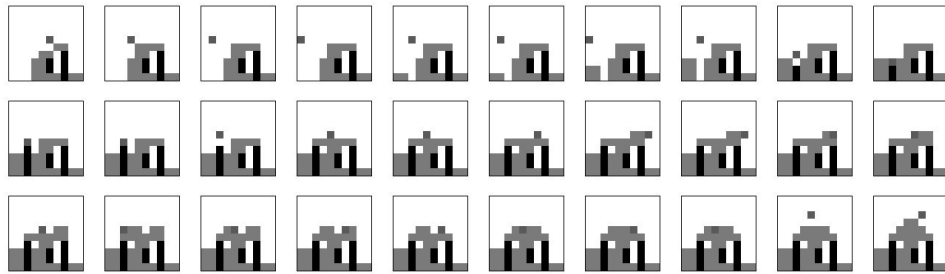
In the following frames of Fig. 7 the assembly plan proceeds to lay scaffolding for what will be the leftmost leg and leftmost cantilever of the arch.

The assembly continues in Fig. 8 as the plan continues to lay bricks that are simultaneously scaffolding for the leftmost cantilever and for the left leg of the arch. Once scaffolding is laid on both sides, both vertical bricks of the left leg



**Fig. 7.** Robust Assembly Plan Steps 1-18: In the first steps, the builder lays scaffolding

are placed. By the final frames of Fig. 8 all the bricks of the final structure are in place.



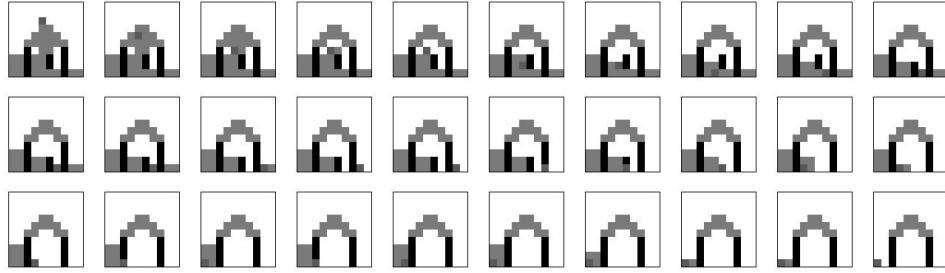
**Fig. 8.** Robust Assembly Plan Frames 19-49: more scaffolding is lain and the arch is completed

All that remains is for the builder to remove the scaffolding, as it does in Fig. 9, leaving, finally, the complete goal structure.

## 4 Conclusion

We have demonstrated that using only evolutionary-scale feedback, ballistic assembly plans can be evolved to overcome a noisy development environment. They are able to do this largely by means of ontogenic scaffolding - intermediate and temporary structural elements necessary for reliable assembly of the goal structure. Our result of an assembly plan capable of 70% yield is typical of our system. Running the evolution for longer can likely result in higher yields, but search grows harder over time as the size of the Pareto-front population increases - a consequence of using multi-objective optimization.

It is worth noting that the assembly of the structure shown in Figs. 7- 9 falls into two distinct ontogenic phases - in the first phase the structure is built with the aid of scaffolding, and in the second, the scaffolding is removed. The presence of two distinct phases, as opposed to a process in which scaffolding is created and



**Fig. 9.** Robust Assembly Plan Frames 50-80: scaffolding is removed

removed for each element of the final structure, is likely due to the specific search gradient created by the two objectives which compare the assembled structure to the goal structure. Evolved assembly plans can first improve along the dimension of missing bricks until they begin to reliably generate all of the parts of the goal structure. Once this is achieved, they can then focus on minimizing the number of extraneous bricks in the structure.

Ontogenic scaffolding, while demonstrably useful, provides a challenge for evolutionary design. To begin with, assembly plans which place and then remove scaffolding will by necessity be longer than those that don't. Secondly, any intermediate assembly plan which places scaffolding but doesn't remove it may incur a penalty for the extraneous structure - the cost of exploration, therefore, tends to be high. Finally, for sufficiently complex structures, the scaffolding itself may require meta-scaffolding. These conditions, among others, combine to make the evolution of ontogenic scaffolding, even in simple environments, a non-trivial task.

Our next step will be to explore methods of evaluating assembly plans in noisy environments without a goal structure provided *a priori*. Without the ability to measure yield, the task is complicated quite a bit. Ultimately, we suspect that more powerful and versatile encodings - such as generative representations [8, 9], or gene-regulatory networks [10], equipped with ontogenic-level feedback, will be better able to adapt to stochastic assembly.

## References

1. Lohn, J., Crawford, J., Globus, A., Hornby, G., Kraus, W., Larchev, G., Pryor, A., Siviastava, D.: Evolvable systems for space applications. In: International Conference on Space Mission Challenges for Information Technology (SMC-IT). (2003)
2. Pollack, J.B., Lipson, H., Hornby, G., Funes, P.: Three generations of automatically designed robots. *Artificial Life* **7** (2001) 215–223
3. Kumar, S., Bentley, P.J.: Computational embryology: past, present and future. In: Advances in evolutionary computing: theory and applications. Springer-Verlag New York, Inc. (2003) 461–477

4. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. *Artificial Life* **9** (2002) 93–130
5. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: The use of simulation in evolutionary robotics. In: *Proc. of the Third European Conference on Artificial Life (ECAL'95)*, Granada, Spain (1995) 704–720
6. Sims, K.: Evolving virtual creatures. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press (1994) 15–22
7. Hornby, G.S.: *Generative Representations for Evolutionary Design Automation*. PhD thesis, Brandeis University, Dept. of Computer Science, Boston, MA, USA (2003)
8. Hornby, G.S., Pollack, J.B.: The advantages of generative grammatical encodings for physical design. In: *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, IEEE Press (2001) 600–607
9. Toussaint, M.: Demonstrating the evolution of complex genetic representations: An evolution of artificial plants. In: *2003 Genetic and Evolutionary Computation Conference (GECCO 2003)*. (2003)
10. Bongard, J.C., Pfeifer, R.: Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E., eds.: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, California, USA, Morgan Kaufmann (2001) 829–836
11. Funes, P.: *Evolution of Complexity in Real-World Domains*. PhD thesis, Brandeis University, Dept. of Computer Science, Boston, MA, USA (2001)
12. Koza, J.R.: *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. MIT Press: Cambridge, MA (1992)
13. Viswanathan, S., Pollack, J.: On the evolvability of replication fidelity in stochastic construction. Technical Report CS-04-248, Brandeis University (2003)
14. Coello, C.A.C.: An updated survey of evolutionary multiobjective optimization techniques: State of the art and future trends. In Angeline, P.J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzal, A., eds.: *Proceedings of the Congress on Evolutionary Computation*. Volume 1., Mayflower Hotel, Washington D.C., USA, IEEE Press (1999) 3–13
15. Fonseca, C.M., Fleming, P.J.: Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: *Genetic Algorithms: Proceedings of the Fifth International Conference*, Morgan Kaufmann (1993) 416–423
16. Winston, H.P.: *Learning By Analyzing Differences*. In: *Artificial Intelligence: Third Edition*. Addison-Wesley, Reading MA (1993) 349–364
17. De Jong, E.D., Watson, R.A., Pollack, J.B.: Reducing bloat and promoting diversity using multi-objective methods. In Spector, L., Goodman, E., Wu, A., Langdon, W., Voigt, H.M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M., Burke, E., eds.: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001*, San Francisco, CA, Morgan Kaufmann Publishers (2001) 11–18
18. Langdon, W.B.: The evolution of size in variable length representations. In: *1998 IEEE International Conference on Evolutionary Computation*, Anchorage, Alaska, USA, IEEE Press (1998) 633–638