Measuring and Optimizing Behavioral Complexity

Faustino J. Gomez, Julian Togelius, and Juergen Schmidhuber

IDSIA, Galleria 2
6928 Manno-Lugano
Switzerland

Abstract. Model complexity is key concern to any artificial learning system due its critical impact on generalization. However, EC research has only focused phenotype structural complexity for static problems. For sequential decision tasks, phenotypes that are very similar in structure, can produce radically different behaviors, and the trade-off between fitness and complexity in this context is not clear. In this paper, behavioral complexity is measured explicitly using compression, and used as a separate objective to be optimized (not as an additional regularization term in a scalar fitness), in order to study this trade-off directly.

1 Introduction

A guiding principle in inductive inference is the concept of parsimony: given a set of competing models that equally explain the data, one should prefer the simplest according to some reasonable measure of complexity. A simpler model is less likely to overfit the data, and will therefore generalize better to new data arising from the same source.

In EC, this principle has been applied to encourage minimal phenotypic structure (e.g. GP programs, neural network topologies) by penalizing the fitness of overly complex individuals so that selection drives the search toward simpler solutions [6, 9, 10, 13].

The advantage of incorporating this *parsimony pressure* has been demonstrated convincingly in supervised learning tasks, producing solutions that are significantly more general. However, for dynamic tasks involving sequential decisions (e.g. reinforcement learning), a phenotype's structural complexity may not be a good predictor of its *behavioral complexity* [5] (i.e. the complexity of the observation-action sequences generated by the evolving policies). Phenotypes that are very similar in structure, can produce radically different behaviors, and the trade-off between fitness and complexity in this context is not clear. In this paper, behavioral complexity is measured explicitly using compression, and used as a separate objective to be optimized (not as an additional regularization term in a scalar fitness), in order to study this trade-off directly.

Multi-Objective approaches have been used previously to control structural complexity (or promote diversity [**?**]), but only in a supervised learning context [2, 3], and always to promote parsimonious solutions. The goal here is to

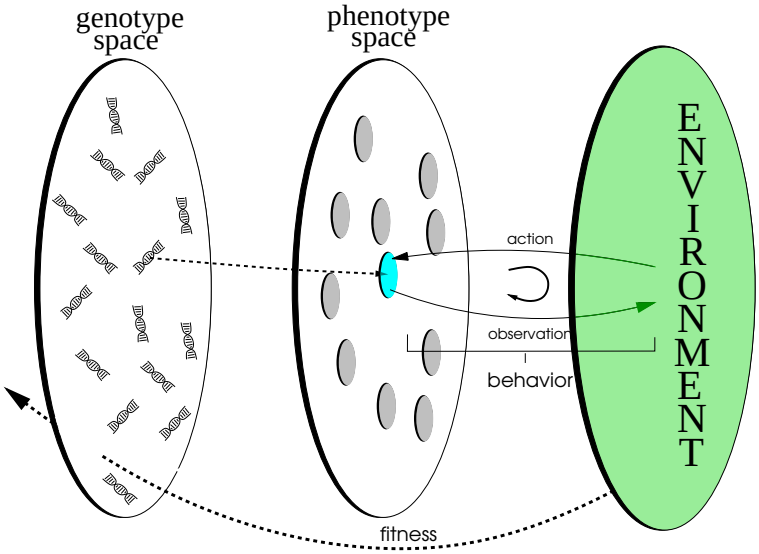

Fig. 1. Genotype-Phenotype map. The complexity of evolving candidate solutions can be computed at different levels. In sequential decision tasks, measuring the structural (model) complexity in phenotype space may not give a reliable indication of the relative complexity of the phenotype behavior (shown as the cycling of actions and observations of the two highlighted phenotypes).

look at complexity more generally, and analyze how encouraging both low *and* high behavioral complexity relates to and can affect performance (fitness) in reinforcement learning tasks.

The next section describes the general idea of complexity within the context of EC. Section 3, presents our experiments in evolving neural network controllers using a multi-objective evolutionary algorithm for two different reinforcement learning tasks: the Tartarus problem, and the Simplerace car driving task. Section 4 provides some analysis of our results and direction for future research.

2 Measuring Complexity

In evolutionary algorithms, the complexity of an individual can be measured in the genotype space where the solutions are encoded as strings, or in the phenotype space where the solutions are manifest.

For some problem classes and genetic representations, measuring complexity in one space is equivalent to applying it in the other: the genotype→phenotype mapping, G, preserves the ordering between objects in each, $complexity(x) > complexity(y) \rightarrow complexity(\hat{x}) > complexity(\hat{y})$, where x, y are genotypes, and $\hat{x}, \hat{y}$ their corresponding genotypes. When this is not the case, it may be more informative to measure the complexity phenotypes (figure 1), after all what we are truly interested in is the complexity of solutions, not there encodings.

For sequential decision tasks (e.g. reinforcement learning), G maps x to some form of policy, π, that implements a probability distribution over a set of possible actions, conditioned on the observation from the environment. More generally, the choice of action at time t can be conditioned on the entire *history* of previous observations, $o \in O$, and actions, $a \in A$:

$$a_t \leftarrow \pi(o_{t-1}, a_{t-1}, \ldots, o_0, a_0). \tag{1}$$

where O is the set of possible observations, and A is the set of possible actions. Such a policy could be implemented, for by a recurrent neural network. In this case, structural complexity can be misleading as policies that are structurally similar with respect to a chosen metric may be very different in terms of behavior when they interact with the environment. We define the *behavior* of individual x to be a set of one or more histories of the form in equation 1 resulting from one or more evaluations in the environment. A behavior is therefore an approximation of the *true* behavior of the individual that can only be sampled by interaction with the environment.

Measuring behavior complexity requires computing a function over the space of possible behaviors for a given $\{A, O\}$. A general framework, rooted in algorithmic information theory [7], that can be used to quantify complexity is the Minimum Description Length Principle [8], which states that any regularity in the data can be used to compress it. Compressing the data means recoding it such that it can be represented using fewer symbols. For a given compression scheme, and two objects (e.g. bit-strings) of the same length, the object whose compressed representation requires the fewest symbols can be considered less complex as it contains more identifiable regularity [1]. In the experiments that follow, this idea is applied to assess the complexity of evolved neural network behaviors using an real-world compressor.

MDL inspired complexity measures have been used in conjunction with evolutionary algorithms before to address *bloat* in Genetic Programming [6] and to evolve minimal neural networks [9, 10, 13], i.e. to control phenotype structural complexity. In the next section, data compressibility is used to measure the complexity of phenotype behaviors, and is used as additional objective to be optimized in order study the interaction between fitness and complexity at the behavioral level.

3 Experiments

To ensure a degree of generality our experiments were conducted in two substantially different reinforcement learning benchmark domains: Tartarus and Simplerace. The three following objectives were used in various combinations:

1. $\boldsymbol{P}$: the standard performance measure or fitness use for the task.
2. $\boldsymbol{C}$: the length of the behavior after applying the Lempel-Ziv [**?**] based `gzip` compressor to it. Behaviors with low C are considered less complex as they contain more regularity for the compressor to exploit.

3. $\boldsymbol{H}$: the Shannon entropy of the behavior: $-\sum p(x_i)log(p(x_i))$, where each x_i is one of the possible symbols representing an action or observation in the behavior. The entropy computes the lower bound on the average number of bits per symbol required to represent the behavior.

Four sets of multi-objective experiments were conducted, each using a different pairing of objectives: $\boldsymbol{M_{PH}}$,$\boldsymbol{M_{P-H}}$,$\boldsymbol{M_{PC}}$,$\boldsymbol{M_{P-C}}$, where the first subscript is the first objective which is always maximized (P in all cases), and the second subscript is the second objective which is maximized, unless it is preceded by a minus sign, in which cased it is minimized. The multi-objective experiments aimed at exploring the interplay between performance, P, and one of the behavioral complexity, C or H.

For the multi-objective experiments, the *NSGA-II* algorithm was used, which is one of the most widely used multi-objective GAs and is known for robust performance in diverse conditions [4]. At each generation, the scores on all three objectives, $\{P, C, H\}$ were recorded for two individuals in the Pareto front: the one that performed best, and the one that had the best score on the chosen complexity-related objective. At the end of each run, all fitnesses were recorded for all individuals in the Pareto front of the final generation, both on the set of problem cases used during evolution and a different set of test cases.

In all of experiments, the controllers were represented by recurrent neural networks (figure 3, details below), and the population size was set to 100. Each run lasted for 4000 generations for the Tartarus problem, and 200 generations for the Simplerace problem. No recombination was used; the only variation operator was mutation, consisting in adding real numbers drawn from a Gaussian distribution with mean 0 and standard deviation 0.1 to all weights in the network.

For both tasks it was not necessary to include observations in the behaviors because the environments are deterministic and the initial states were fixed for all individuals in a single run, so that each sequence of actions only has one corresponding sequence of observations.

3.1 The Tartarus Problem

Figure 2a describes the Tartarus problem [11], used in the experiments. Although the grid-world is quite small, the task is challenging because the bulldozer can only see the adjacent grid cells, so that many observations that require different actions look the same, i.e. perceptual aliasing. In order to perform the task successfully, the bulldozer must remember previous observations such that it can compute its location relative to the walls and record the locations of observed blocks for the purpose quickly acquiring them later. In short, the agent is quite blind which means that evolutionary search can quickly discover simple, mechanical behaviors that produce better than random performance but do not exhibit the underlying memory capability to perform well on the task.

The Tartarus controllers were represented by fully recurrent neural networks with five sigmoidal neurons (figure 3a). Each controller was evaluated on 100

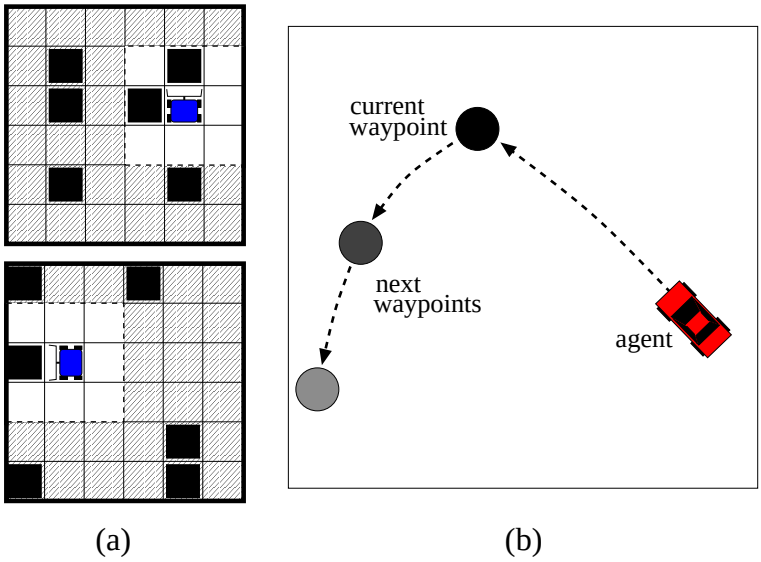

Fig. 2. The (a) Tartarus and (b) Simplerace tasks. The upper Tartarus board shows a possible initial state with the six blocks and the bulldozer placed at random squares away from the walls; the orientation the bulldozer is also random. The bulldozer must select an action (either turn left, turn right, or go forward) at each time-step based on the situation within its visual field (shown in white), and its internal state (memory). The bulldozer can only move forward if its path is unobstructed or the block in its way has no block behind it, otherwise it will remain its current position. The lower board is a possible final state after the alloted 80 moves. The score for this configuration is 7: two blocks receive a score of two for being in the corner, plus one point for the other three blocks that are against a wall. The object is the drive the car (both accelerator and steering) through as many randomly place waypoints in an alloted amount of time.

random board configurations. To reduce evaluation noise, the set of 100 initial boards was chosen at random for each simulation, but remained fixed for the duration of the simulation. That is, in a given run all networks were evaluated on the same 100 initial boards. The behaviors consisted of sequences of 80 {Left=1, Right=2, Forward=3} actions executed in each of the 100 trials conc

3.2 Simulated Race Car Driving

The *simplerace* problem involves driving a car in a simple racing simulation in order to reach as many randomly placed waypoints as possible in a limited amount of time (figure 2b). There are plenty of good controllers to compare our results with, as the game has previously been used as a benchmark problem in several papers, and in two competitions associated with recent conferences[1].

[1] A description of the problem is available in [12], and source code can be downloaded from http://julian.togelius.com/cec2007competition.

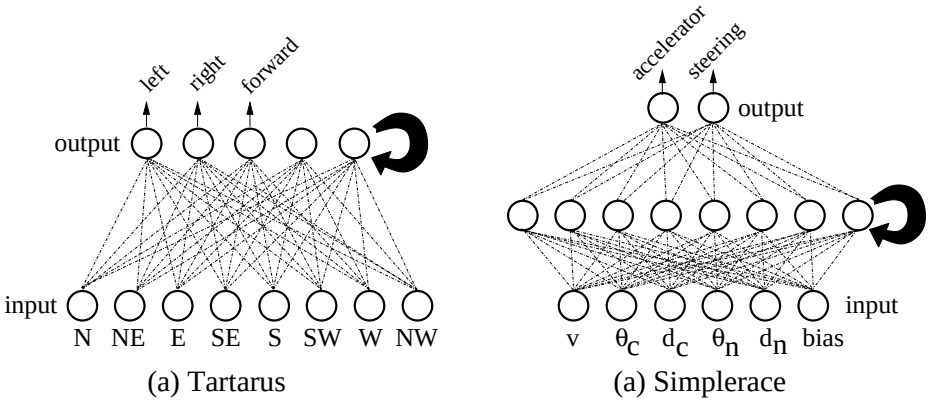

Fig. 3. (a) Tartarus and (b) Simplerace controllers. The Tartarus "bulldozer" is controlled by a fully recurrent neural network (the recurrent connections denoted by the large black arrow) with five units. At each time step the network outputs the action corresponding to action unit (left, right, forward) with the highest activation based on the state of the eight surrounding grid cells. The Simplerace car is controlled by a simple recurrent network with eight hidden units; v, speed of the car, θ_c, and d_c, the angle and distance to the current waypoint, θ_n, and d_n, the angle and distance to the next waypoint.

The Simplerace controllers were represented by simple recurrent networks (SRN; figure 3b) with six inputs, eight hidden sigmoidal units, and two outputs. The inputs consisted of: (1) the speed of the car, (2) the angle and (3) distance to the current, the (4) angle and (5) distance to the next way point, and (6) a bias term. The two output units are used to represent nine actions using the following scheme: the first unit steers the car, an activation of < -0.3 means "turn left", between -0.3 and 0.3 means "go straight", and > 0.3, means "turn right". The second unit controls the forward-backward motion, < -0.3 means "go forward", between -0.3 and 0.3 means "put the car in neutral", and > 0.3, means "brake" (if the car is no longer moving forward this action puts the car in reverse). Each network was evaluated using the same set of 10 cases (i.e. waypoint locations) each lasting 1000 time-steps (actions), chosen at random at the beginning of each simulation.

3.3 Results

Figure 4 shows the performance, P, for the four multi-objective configurations. The "high complexity" configurations, M_{PH} and M_{PC}, performed significantly better than the "low complexity", M_{P-H} and M_{P-C}, on both tasks, but the effect was more pronounced for the Simplerace task where minimizing H interferes strongly with fitness. The problem with selecting for low entropy solutions in Simplerace may be that, because the task has nine actions (compared with 3 for Tartarus), entropy can be reduced greatly by restricting the number of

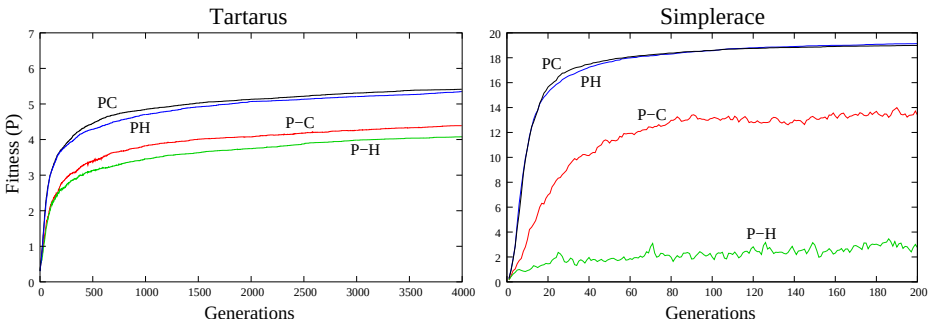

Fig. 4. Performance on Tartarus and Simplerace. Each curve denotes the fitness of the best individual in each generation for each of the four multi-objective configurations. Average of 50 runs.

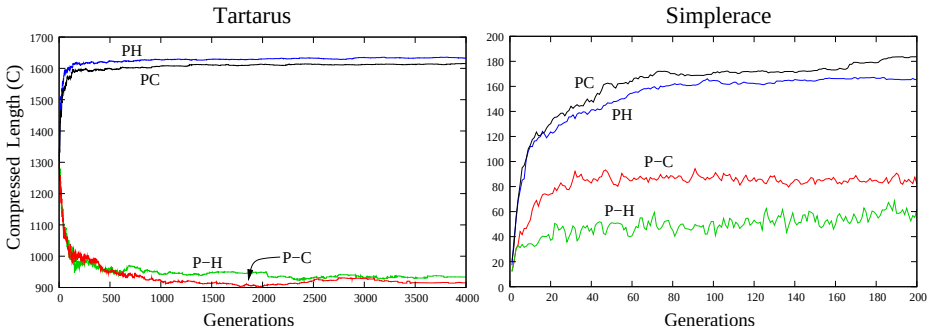

Fig. 5. Compressed Length. Each curve shows, for the each configuration, the compressed length of the most fit individual from each generation. Average of 50 runs.

actions used, whereas compression can work by forcing the sequence of actions into regular patterns that still utilize all actions.

The difference between M_{PC} and M_{PH} on both tasks was not statistically significant. Pushing complexity, either by maximizing C or H, promotes policies that make more full use of their action repertoire. As there are many more high complexity sequences than low complexity sequences (i.e. low complexity sequences tend to be more similar), diversity in the population is better maintained allowing evolutionary search to discover more fit solutions.

Figure 5 shows the compressed length (C) of the most fit individual in the population for the four configurations. Here, again, there is a clear distinction between the maximization and minimization runs, as should be expected, but the two tasks have very different regimes. In both, maximizing complexity (C *or* H) increases the compressed length of the most fit individual. For Tartarus the C of the most fit individual starts at an intermediate value of around 1300, and then rises or drops sharply until reaching a steady value for rest of the run. In contrast, the Simplerace runs always start with very compressible behaviors,

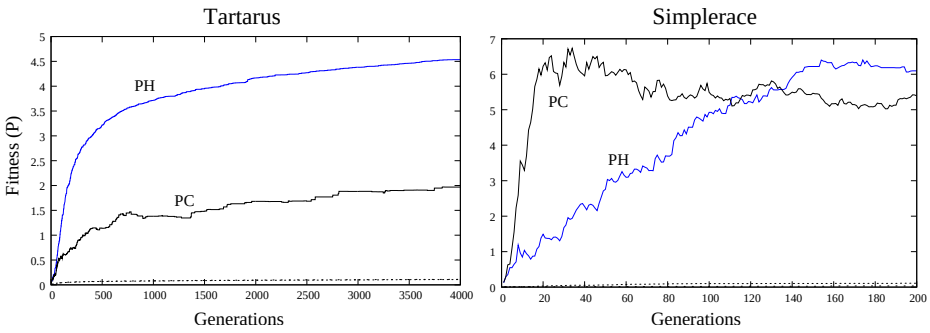

Fig. 6. Fitness of Most/Least complex individual. Each curve shows, for the each configuration, the fitness score of the individual in each generation with the best complexity (highest or lowest, depending on whether it is being maximized or minimized), in terms of either C or H, depending on which is being used as the second objective. The minimization runs are at the bottom of graph and are indistinguishable. Average of 50 runs.

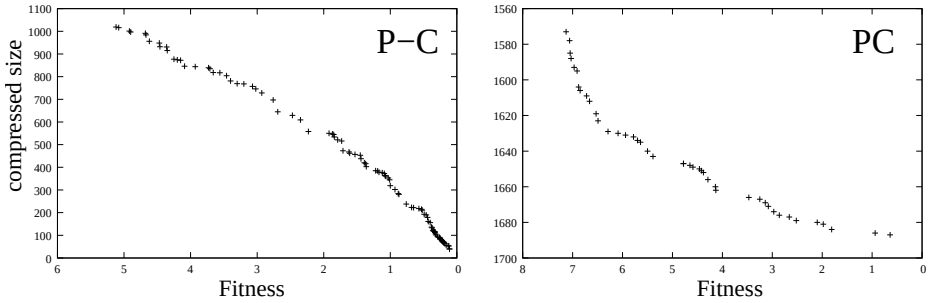

Fig. 7. Pareto Front: fitness/complexity trade-off. The plot on the left show a typical final Pareto front for M_{P-C}. When behavioral complexity as measured by compressed length is minimized, high fitness ($P > 6$) is not achieved, with many When behavioral complexity is maximized, M_{PC}, the range of complexity values is high.

which gradually become more complex, even for the minimization configurations. The reason for this is very likely due, at least in part, to the output representation used in the Simplerace network. Because each output unit can select one of three actions (as opposed the one-action-per-unit scheme for Tartarus), the initial random networks will tend to have units that saturate at 0 or 1 such that the behaviors will have very low complexity.

Figure 6 shows the fitness of the most complex individual (either in terms of C or H, depending on the measure being optimized). For Tartarus, the most complex individual in M_{PC} is less correlated with fitness compared to M_{PH}, suggesting that is P are C conflict more than P and H. For Simplerace, the fitness of the least compressible behavior increases rapidly and then gradually

trends downward, whereas the fitness of the behavior with the highest entropy rises steadily throughout the run.

Figure 7 shows the Pareto fronts of the final generation of a typical M_{PC} and M_{P-C} run for Tartarus (Simplerace produces very similar results, also for the PH runs). For M_{P-C} the most fit non-dominated individuals are also the most complex, with most solutions concentrated around the lowest complexity. For M_{PC}, the the complexity is in a much higher range (note the y-axis is inverted w.r.t. fig (a)), and, in contrast with M_{P-C}, the most fit solutions are the *least* complex. So while parsimony is favorable for given level of fitness, suppressing complexity from the outset, as in M_{P-C}, works against acquiring high fitness (compare the max fitness in figure 4).

4 Conclusions

This paper constitutes a preliminary study of the evolutionary trade-off between fitness and complexity. The results, most clearly illustrated by the final Pareto fronts in figure 7, are consistent with heuristic shaping techniques used in supervised learning where model complexity is given a lower priority in the early stages of learning so that the learner acquires more degrees of freedom with which to reduce error. Once, the error reaches a set threshold, the complexity of the model is penalized to reduce the number of free parameters and in order to improve generalization [**?**].

The relationship between entropy and Lempel-Ziv (e.g. `gzip`) is complex. Entropy is only concerned with the expected occurrence of each symbol in the behavior, not the ordering or structure of the behavior. The compressor also relies on entropy to encode the behavior, but only after analyzing the structure of the symbol sequence. While entropy and algorithmic complexity are asymptotically equivalent, two strings with equal entropy can have very different compressibility due to structure. In our experiments, these differences produced very different dynamics (see figures 5 and 6), but very similar fitness. The overriding effect seems to be that of increasing diversity, as discussed in section 3.3.

Further analysis is required to measure, e.g. the correlation between complexity and generalization for a given fitness level to determine whether less complex solutions with are more robust. And the behaviors themselves should be analyzed to see if qualitatively different policies arise when complexity of driven in terms of entropy, gzip, or other compressors (e.g. PPM, bzip2) that exploit different algorithmic regularities.

For sequential decision tasks, behavior seems to be the right level at which to compare individuals [5], but, of course, model complexity is critical in determining the range of possible behaviors available to the agent. Future work will also look at combining structural and behavioral complexity criteria for evolutionary methods that search, e.g. both neural network topology and weight space.

Acknowledgments

This research was supported in part by the EU Projects IM-CLEVER (#231711), STIFF (#231576), Humanobs (#231453), and the NSF under grant EIA-0303609.

References

1. A. Baronchelli, E. Caglioti, and V. Loreto. Artificial sequences and complexity measures. *Journal of Statistical Mechanics*, 2005.
2. E. D. De Jong and J. B. Pollack. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines*, 4(3):211–233, 2003.
3. E. D. De Jong, R. A. Watson, and J. B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18. San Francisco, CA: Morgan Kaufmann, 2001.
4. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2002.
5. F. Gomez. Sustaining diversity using behavioral information distance. In *Proceedings of the Genetic and Evolutionary Computation Conference (to appear)*, 2009.
6. H. Iba, H. D. Garis, and T. Sato. Genetic programming using a minimum description length principle. In *Advances in Genetic Programming*, pages 265–284. MIT Press, 1994.
7. M. Li and P. M. B. Vitányi. An introduction to Kolmogorov complexity and its applications. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 188–254. Elsevier Science Publishers B.V., 1990.
8. J. Rissanen. Modeling by shortest data description. *Automatica*, pages 465–471, 1978.
9. B. tak Zhang and H. Muhlenbein. Evolving optimal neural networks using genetic algorithms with occam's razor. *Complex Systems*, 7:199–220, 1993.
10. B. tak Zhang and H. Muhlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3:17–38, 1995.
11. A. Teller. *Advances in Genetic Programming*, chapter 9. MIT Press, 1994.
12. J. Togelius. *Optimization, Imitation and Innovation: Computational Intelligence and Games*. PhD thesis, Department of Computing and Electronic Systems, University of Essex, Colchester, UK, 2007.
13. B.-T. Zhang, P. Ohm, and H. Mühlenbein. Evolutionary induction of sparse neural trees. *Evolutionary Computation*, 5(2):213–236, 1997.