Incremental Evolution of HERCL Programs for Robust Control

Alan Blair
School of Computer Science and Engineering
University of New South Wales
Sydney, 2052, Australia
blair@cse.unsw.edu.au

ABSTRACT

We explore the evolution of programs for control tasks using the recently introduced Hierarchical Evolutionary Re-Combination Language (HERCL) which has been designed as an austere and general-purpose language, with a view toward modular evolutionary computation, combining elements from Linear GP with stack-based operations from FORTH. We show that HERCL programs can be evolved to robustly perform a benchmark double pole balancing task from a specified range of initial conditions, with the poles remaining balanced for up to an hour of simulated time.

Categories and Subject Descriptors

ALIFE [**Aritificial Life/Robotics/Evolvable Hardware**]: Evolution of controllers; GP [**Genetic Programming**]: linear GP, stack-based GP

Keywords

control; linear GP; stack-based GP; modular evolution

1. INTRODUCTION

The Hierarchicial Evolutionary Re-Combination paradigm and associated HERCL programming language were recently introduced [1] in an effort to design a novel framework for evolutionary automatic programming with certain key requirements, hypothesized to be advantageous for the development of modular evolving systems.

HERCL programs have previously been evolved to perform coding tasks such as the Caesar and Vigenere cipher [1]. In the present work, we explore the evolution of HERCL programs for dynamically unstable control problems, using the benchmark double pole balancing task as a case study.

2. HIERARCHICAL EVOLUTION

HERCL has been designed with the lofty goal of eventually building a Society of Mind model of cognition [3] in which a heterogenous collection of agents (modules) pass messages to each other and work together to achieve an overall task.

Ideally, such a system should improve itself through a process of modular evolution, with the various agents within the system all evolving simultaneously — each agent competing against other agents in its own ecological *niche*. In contrast to *holistic* evaluation — where variations in individual components are judged by re-testing the fitness of the entire system — we are instead interested in exploring *peer* evaluation, where each component is evaluated by other components.

The *global* interactions between different agents in this kind of system would raise formidable challenges and complexities, in terms of collusion, deception and evolutionary dynamics. HERCL represents a modest first step, in trying to identify certain essential requirements which should theoretically be satisfied *locally* by the evolutionary process in each individual niche, and attempting to design (and refine) an appropriate framework, and language, to meet these requirements:

1) Each agent should take the form of a (genetic) program, in a language which is austere and flexible enough to handle a wide variety of computational tasks.
2) Agents should communicate by passing messages to each other in a general-purpose format.
3) The number of agents competing in each ecological niche should be relatively small (so as not to put an excessive burden of evaluation on other agents in the system) but still able to search broadly and escape from local optima.

HERCL does not use a population as such, but instead maintains a stack or *ladder* of candidate solutions (agents), and a *codebank* of potential mates (Figure 1).

At each step of the algorithm, we select the agent at the top rung of the ladder and apply either mutation or crossover with a randomly chosen agent from the codebank. HERCL programs are divided hierarchically into cells, bars, instructions and codons. We distinguish different *levels* of mutation/crossover (TUNE, POINT, BAR, BRANCH, CELL, etc.) which vary according to the portion of code from the primary (ladder) parent that is either mutated or crossed over with a commensurate portion of code from the secondary (codebank) parent. The level of each mutation/crossover is chosen randomly, with lower levels weighted more heavily than higher ones, and with the constraint that the mutation levels must strictly decrease as we move up the ladder. The agents in the codebank are grouped according to muta-

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
ACM 978-1-4503-2881-4/14/07.
http://dx.doi.org/10.1145/2598394.2598424.

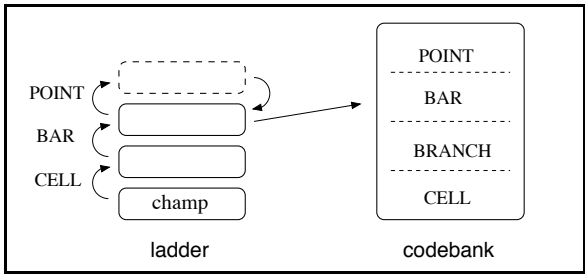

Figure 1: Hierarchical Evolutionary Re-Combination. If the top agent on the ladder becomes fitter than the one below it, the top agent will move down to replace the lower agent (which is transferred to the codebank). If the top agent exceeds its maximum number of allowable offspring without ever becoming fitter than the one below it, the top agent is removed from the ladder (and transferred to the codebank).

tion/crossover level, with a limited number of agents in each level. Further details can be found in [1].

3. DOUBLE POLE BALANCING

We adopt the benchmark double pole balancing task for this case study, and try as far as possible to use the same scenario and parameter settings as in previous works [2, 4, 5, 6]. Two poles (inverted pendula) of length 1m and 0.1m are attached by hinges to a wheeled cart running on a track, and an agent must apply a positive or negative force at each timestep in order to keep the poles from falling over, and keep the cart from runnning off the track. The equations of motion for this system can be found in [6]. At each timestep (0.02 simulated seconds) the agent is presented with the position x and velocity $\dot{x}$ of the cart as well as the angles θ_1, θ_2 and angular velocity $\dot{\theta}_1, \dot{\theta}_2$ of the poles, and must choose to apply a force of either $+10$N or -10N (bang-bang control). The system is considered balanced so long as x remains within ± 2.4m and θ_1, θ_2 within ± 0.6 radians ($35°$).

In contrast to previous approaches — where the controllers were evolved on a fixed initial configuration (IC) and then tested for generalization on other IC's, to a maximum of 1000 timesteps — we are instead interested in the possibility of evolving a *robust* controller, which is evolved and tested on a range of IC's, but is required to achieve nearly 100% success within this range, and keep the poles balanced for a much longer time period.

In order to achieve this robustness, we follow a process of incremental training, where a new IC is added to the fitness evaluation whenever the best agent (champ) has achieved *success* for all the current configurations (by keeping the two poles balanced for 50,000 timesteps).

Ten evolutionary runs were performed with different random seeds. In each case, the first agent to solve 1, 10, 100 and 1000 IC's were extracted, and tested on 1000 newly generated IC's, chosen randomly within these bounds:

$$x \in [-1, 1], \dot{x} \in [-0.1, 0.1], \theta_{1,2} \in [-0.05, 0.05], \dot{\theta}_{1,2} \in [-0.01, 0.01]$$

Figure 2 illustrates the robustness (generalization) of the extracted agents, for each of the ten evolutionary runs. We see that a significant gain in robustness is achieved by extending the evolution to 10 training configurations rather

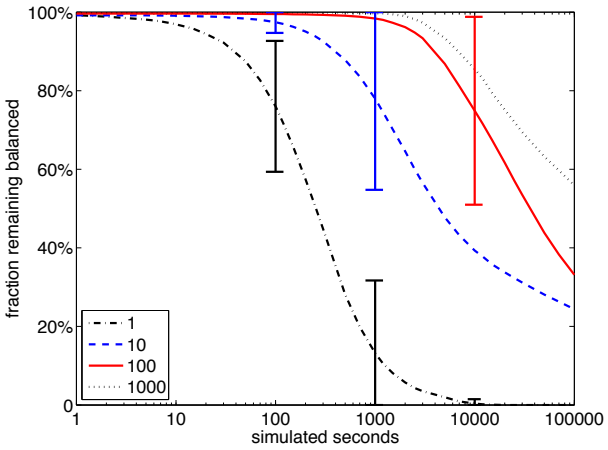

Figure 2: Robustness (generalization) for agents evolved on 1, 10, 100 and 1000 (training) IC's. Each plot is a mean of ten runs (with error bars shown selectively) showing simulated seconds (log scale) vs. fraction of (test) IC's for which the two poles remain balanced.

than stopping at one. On average, this increases the chance of remaining balanced for 50,000 timesteps (1000 seconds) from 13% to 78%. Two of the runs eventually produced agents which can keep both poles balanced, for the full set of 1000 test IC's, for 5 million timesteps (equivalent to 100,000 seconds, or slightly more than a day of simulated time).

4. CONCLUSION

We have shown that HERCL programs can be evolved to robustly perform the benchmark double pole balancing task from a defined range of initial conditions, and keep the poles balanced for an hour or more of simulated time. Incremental training can significantly improve the robustness of the solution, but eventually there is a law of diminishing returns due to long evaluation times.

These results extend the range of computational tasks amenable to Hierarchical Evolutionary Re-Combination, and provide further evidence that the HERCL framework may possess the versitility required to build modular evolving systems, which is the subject of future work.

5. REFERENCES

[1] Blair, A., 2013. Learning the Caesar and Vigenere Cipher by Hierarchical Evolutionary Re-Combination, *2013 Congress on Evolutionary Computation*, 605–612.

[2] Gomez, F. and R. Miikkulainen, 2002. "Solving non-Markovian Control Tasks with Neuroevolution", *16th Int'l Jount Conf. on Artificial Intelligence*, 1356–1361.

[3] Minsky, Marvin. 1988. *Society of mind*, Simon & Schuster.

[4] Saravanan, N. and D.B. Fogel, 1995. "Evolving neural control systems", *IEEE Expert* 10(3), 23–27.

[5] Stanley, K.O. and R. Miikkulainen, 2002. "Evolving neural networks through augmenting topologies", *Evolutionary Computation* 10(2), 99–127.

[6] Wieland, A., 1991. "Evolving Neural Network Controllers for Unstable Systems", *Int'l Joint Conf. on Neural Networks*, 667–673.