Citation: Clune J, Beckmann BE, Pennock RT, and Ofria C (2009)
HybrID: A hybridization of indirect and direct encodings for evolutionary computation.
Proceedings of the European Conference on Artificial Life. Pages 1-8. Budapest, Hungary.

HybrID: A Hybridization of Indirect and Direct Encodings for Evolutionary Computation

Jeff Clune[1], Benjamin E. Beckmann[1], Robert T. Pennock[1,2] and Charles Ofria[1]

[1] Department of Computer Science and Engineering
[2] Department of Philosophy and Lyman Briggs College
Michigan State University, East Lansing, MI, USA
{jclune, beckma24, pennock5, ofria}@msu.edu

Abstract. Evolutionary algorithms typically use *direct encodings*, where each element of the phenotype is specified independently in the genotype. Because direct encodings have difficulty evolving modular and symmetric phenotypes, some researchers use indirect encodings, wherein one genomic element can influence multiple parts of a phenotype. We have previously shown that HyperNEAT, an indirect encoding, outperforms FT-NEAT, a direct-encoding control, on many problems, especially as the regularity of the problem increases. However, HyperNEAT is no panacea; it had difficulty accounting for irregularities in problems. In this paper, we propose a new algorithm, a Hybridized Indirect and Direct encoding (HybrID), which discovers the regularity of a problem with an indirect encoding and accounts for irregularities via a direct encoding. In three different problem domains, HybrID outperforms HyperNEAT in most situations, with performance improvements as large as 40%. Our work suggests that hybridizing indirect and direct encodings can be an effective way to improve the performance of evolutionary algorithms.

Keywords: Indirect (generative, developmental) encodings (representations), artificial neural networks, neuroevolution, evolutionary algorithms.

1 Introduction

Evolutionary algorithms, such as neuroevolution and genetic algorithms, typically use *direct encodings*, where each element of a phenotype is independently specified in its genotype. However, these direct encodings are limited in their ability to evolve complex, modular, and symmetric phenotypes because individual mutations cannot produce coordinated changes to multiple elements of a phenotype [1]. Such coordinated mutational effects can occur with *indirect encodings*, also called *developmental* or *generative* encodings, wherein a single element in a genotype can influence many parts of the phenotype [1, 2]. Indirect encodings have been shown to produce highly regular solutions to problems [1, 3-5], but their bias toward regularity makes it difficult for them to properly handle irregularities in problems [4].

In this paper, we propose a new algorithm that is a Hybridized Indirect and Direct encoding (HybrID), which combines the benefits of both encodings. Although we

present one specific implementation of a HybrID, we apply the term to any combination of indirect and direct encodings. While we are not aware of any prior work that specifically combines direct and indirect encodings, researchers have previously altered representations during evolutionary search, primarily to change the precision of values being evolved by genetic algorithms [6]. Other researchers have employed non-evolutionary optimization techniques to fine-tune the details of evolved solutions [7]. However, such techniques do not leverage the benefits of indirect encodings.

This paper presents results from problems where an indirect encoding has already been shown to outperform a direct encoding [3, 4], and demonstrates that HybrID can further improve performance by as much as 40%. The major contribution of this paper is to introduce HybrID as a type of evolutionary algorithm that can both exploit a problem's regularities and account for its irregularities.

2 The Indirect and Direct Encodings and their Hybridization

The HybrID in this paper first runs HyperNEAT [8], an indirect encoding for evolving artificial neural networks (ANNs), and then *switches* to FT-NEAT, its direct-encoding control [3, 8]. We next describe each encoding and their hybridization.

2.1 HyperNEAT, the Indirect Encoding

HyperNEAT is an indirect encoding for evolving ANNs that is inspired by the way natural organisms develop [8]. It evolves Compositional Pattern Producing Networks (CPPNs) [9], each of which is a genome that encodes an ANN phenotype (also called a *substrate*) [8]. Each CPPN is itself a directed graph, where each node is a mathematical function, such as sine or Gaussian. The nature of these functions can facilitate the evolution of properties such as symmetry (e.g., an absolute value or Gaussian function) and repetition (e.g., a sine function) [8, 9]. The signal on each link in the CPPN is multiplied by that link's weight, which can alter its effect.

A CPPN is queried once for each link in the ANN substrate to determine that link's weight. The inputs to the CPPN are the Cartesian coordinates of both the *source* (e.g., $[x_1=2, y_1=3]$) and *target* (e.g., $[x_2=1, y_2=5]$) nodes of a link and a constant bias value. The CPPN takes these five values as inputs and produces one or two output values, depending on the substrate topology. If there is no hidden layer in the substrate, the single output is the weight of the link between a source node on the input layer and a target node on the output layer. If there is a hidden layer, the first output value determines the weight of the link between the associated input (source) and hidden layer (target) nodes, and the second output value determines the weight of the link between the associated hidden (source) and output (target) layer nodes. All pairwise combinations of source and target nodes are iteratively passed as inputs to a CPPN to determine the weight of each substrate link.

HyperNEAT is capable of exploiting the geometry of a problem [8]. Because the link values between nodes in the final ANN substrate are a function of the geometric positions of those nodes, HyperNEAT can exploit such information when solving a

problem [8, 10]. In the case of quadruped locomotion, this property helped HyperNEAT produce gaits with front-back, left-right, and four-way symmetries [3, 10].

The evolution of the population of CPPNs occurs according to the principles of the NeuroEvolution of Augmenting Topologies (NEAT) algorithm [11], which was originally designed to evolve ANNs. NEAT can be fruitfully applied to CPPNs because of their structural similarity to ANNs. For example, mutations can add a node, and thus a function, to a CPPN graph, or change its link weights. The NEAT algorithm is unique in three main ways [11]. Initially, it starts with small genomes that encode simple networks and slowly *complexifies* them via mutations that add nodes and links to the network, enabling the algorithm to evolve the network topology in addition to its weights. Secondly, NEAT has a fitness sharing mechanism that preserves diversity in the system and gives time for new innovations to be tuned by evolution before competing them against more adapted rivals. Finally, NEAT tracks historical information to perform intelligent crossover while avoiding the need for expensive topological analysis. A full explanation of NEAT can be found in [11].

2.2 FT-NEAT, a Direct-Encoding Control

To isolate the effects of changing only the representation, the direct encoding switched to after the HyperNEAT stage is *Fixed Topology NEAT* (FT-NEAT) [3]. FT-NEAT directly evolves each weight in the ANN independently and does not use complexification. All other elements from NEAT (e.g., its crossover and diversity preservation mechanisms) remain the same between HyperNEAT and FT-NEAT.

2.3 A HybrID of HyperNEAT and FT-NEAT

The HybrID presented in this paper runs HyperNEAT for a fixed number of generations and then the encoding is changed to FT-NEAT at a *switch point*. To switch, we transfer the ANN phenotypes of each individual in the HyperNEAT population to FT-NEAT genomes that are then further evolved with FT-NEAT. In the discussion section we describe alternate HybrID instantiations.

3 Results From Three Problem Domains

We compare HybrID to controls on three problems that have scalable regularity to assess HybrID's performance on versions of the same problem with varying amounts of problem regularity. The first two problems were chosen because they are easy to conceptualize, and the third is a more challenging, realistic problem.

We conducted 50 runs of each experimental treatment in this paper, and all data plotted is averaged across them. The parameter configurations for all experiments are similar to those in [3, 4, 8, 10], and can be viewed at http://devolab.msu.edu/SupportDocs/Hybrid. The mutation rate per link was 0.08 for HyperNEAT and 0.0008 for FT-NEAT; preliminary experiments revealed these to be effective mutation rates for each encoding. FT-NEAT has a lower per-link mutation rate because its genome has

many more mutational targets than HyperNEAT. Additional experiments showed no statistically significant increase in HyperNEAT's performance on all three problems when its mutation rate was dropped to 0.0008 at the switch point.

The Target Weights Problem: We begin our analysis with the simple test problem of evolving a specific target ANN. At the start of each run, a target weight is assigned to each link in an ANN, and during the run fitness values are set in proportion to each organism's similarity to that target ANN. As in previous work [4], we scaled the regularity of this problem by varying the percentage of links in the target ANN that were the same, randomly chosen value (Q), from 0% to 100%. All non-Q target weights were each independently assigned a random value. Target weights were in the range [-3, 3]. The ANN had 9 input and 9 output nodes, and was fully connected. HybrID switched from HyperNEAT to FT-NEAT at 100 generations, and the experiment lasted a total of 1000 generations. The population size was 1000.

As previously shown [4], HyperNEAT quickly discovered the regularity in the more regular versions of this problem, but had difficulty making exceptions to account for irregularities, even after hundreds of generations (Fig. 1). FT-NEAT, on the other hand, was slower, but eventually performed well, in part because the problem has no epistatic interactions and thus coordinated mutational effects are not required. HybrID combined the best attributes of both encodings: it quickly discovered the regularity of the problem and, after the encoding switch, was able to further optimize solutions by accounting for irregularities. While HybrID and FT-NEAT eventually evolved solutions of similar quality, early on HybrID did better on more regular problems and less well on less regular problems. HybrID significantly outperformed both HyperNEAT and FT-NEAT at generation 250 on the 70%, 80%, and 90% regular problems ($p < .01$, Wilcoxon rank-sum test). Earlier switch points further improved the speed at which HybrID made progress on this problem (data not shown).

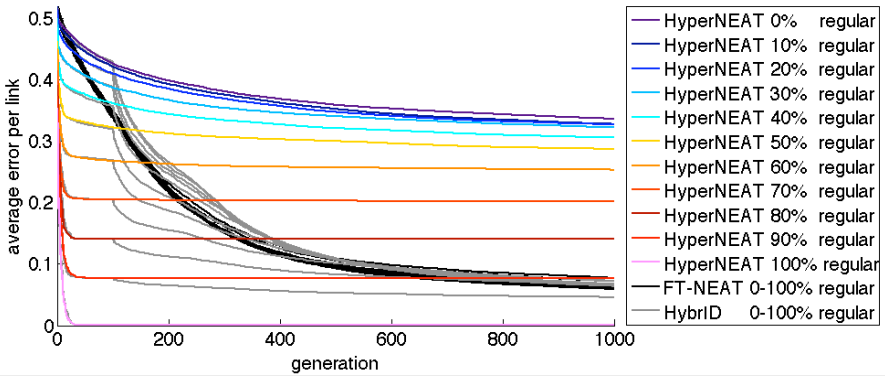

Fig. 1. A comparison of HyperNEAT, FT-NEAT and HybrID on a range of problem regularities for the target weights problem. For each regularity level, a HybrID line (gray) departs from the corresponding HyperNEAT line (colored) at the switch point (generation 100). The performance of FT-NEAT (black lines) was unaffected by the regularity of the problem, which is why the lines are overlaid and indistinguishable. HybrID outperforms HyperNEAT and FT-NEAT in early generations on versions of the problem that are mostly regular but have some irregularities.

The Bit Mirroring Problem: The next problem, called *bit mirroring*, [4] is more challenging and realistic because it has epistasis. In this problem, each input node is paired with a target output node, and the goal is to pass a value from each input node to its associated target output node. The ideal solution features a positive-weight link between each input-output pair, and a zero weight for every other link. The ANN substrate for this experiment had a 49-node input sheet and 49-node output sheet, each arranged in a 7x7 grid. Each input node was connected to all output nodes, totaling 2401 links. The most regular version of the problem paired each input node with the output node in the same row and column. We decreased the regularity, first by reducing the fraction of inputs that were constrained to be in the same *column* as their target, and then by further reducing the fraction of inputs that were constrained to be in the same *row* as their target. Because it has previously been shown that HyperNEAT outperforms FT-NEAT on all versions of this problem [4], here we only compare HybrID to HyperNEAT. A population of size 500 was evolved for 5000 generations and the switch point for HybrID was at generation 2500.

The results reveal that HybrID ties HyperNEAT on the most regular versions of this problem, and provides a significant fitness improvement over HyperNEAT on all versions of the problem that have a certain amount of irregularity (Fig. 2). HybrID's advantage over HyperNEAT was largest on problems of intermediate regularity.

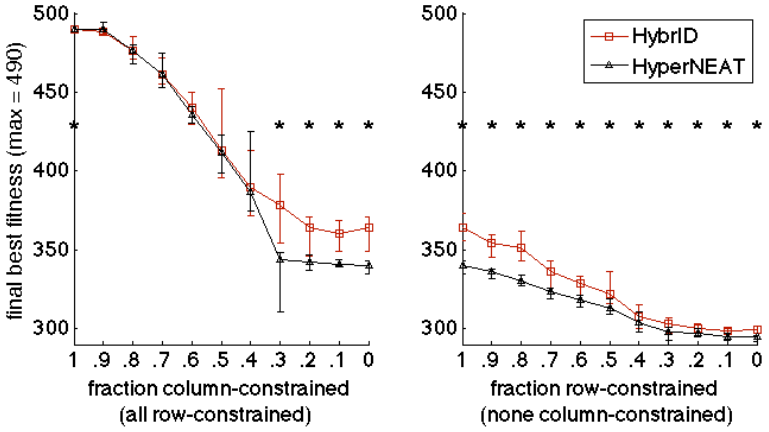

Fig. 2. HybrID vs. HyperNEAT performance on versions of the bit mirroring problem with regularity decreasing from left to right. Plotted are median values ± the 25th and 75th quartiles. Asterisks indicate $p < .05$ (Wilcoxon rank-sum test).

The Simulated Quadruped Controller Problem: The previous two test problems were chosen because they are easy to conceptualize, their regularity can be scaled, and because the ideal solution is known *a priori*. However, it is also important to demonstrate that HybrID succeeds on more complicated, real-world problems, such as evolving controllers for legged robots. HyperNEAT has previously been shown to evolve fast, natural gaits for simulated legged robots [3]. The evolved gaits, however, were extremely coordinated, with all legs often moving in near perfect synchrony. We tested HybrID on this problem to determine if it would improve fitness by facilitating the fine-tuning of aspects of the controller, such as the movements of individual legs.

We repeated the experiment from [3] with HyperNEAT and HybrID, except that half the simulation time was allotted per evaluation due to computational limitations. The ANN substrate consisted of three 5x4 Cartesian grids of nodes forming input, hidden, and output layers. Adjacent layers were completely connected, meaning that there were $(5x4)^2 \times 2 = 800$ links in each substrate. The inputs to the substrate were the current angles of each of the 12 joints of the robot (described below), a touch sensor that provides a 1 if the lower leg is touching the ground and a 0 if it is not, the pitch, roll, and yaw of the torso, and a modified sine wave (to facilitate the production of periodic behaviors). The sine wave was the following function of time (t) in milliseconds: $\sin(120 \times t) \times \pi$. This function produces numbers from $-\pi$ to π, which was the range of the unconstrained joints. During preliminary tests, we experimentally found the constant 120 to produce fast, natural gaits, and determined that the touch, pitch, roll, yaw, and sine inputs all contributed to the ability to evolve fast gaits [3].

The ANN substrate outputs were the desired angles for each joint, which were fed into proportional controllers that applied forces to move the joints toward the desired angles. Robots were evaluated in the ODE physics simulator (www.ode.org). The rectangular torso of the robot was (in ODE units) 0.15 wide, 0.3 long, and .05 tall. Each of four legs was composed of three cylinders (length 0.075, radius 0.02) and three hinge joints. The first cylinder functioned as a hip bone. It was parallel to the proximal-distal axis of the torso and barely stuck out from it. The other two cylinders were the upper and lower leg. There were two hip joints and one knee joint. The first hip joint allowed the legs to swing forward and backward (anterior-posterior) and was constrained to 180° such that, at maximum extension, it was parallel with the torso. The second hip joint allowed a leg to swing in and out (proximal-distal). Together, the two hip joints approximated a universal joint. The knee joint swung forward and backward. The second hip and knee joints were unconstrained.

Each controller in a population of 150 was simulated for 3000 time steps (3 seconds). Experiments lasted 1000 generations with a switch point at generation 500. Trials were cut short if any part of the robot except its lower leg touched the ground, or if the number of direction changes in joints exceeded 960. The latter condition reflects the fact that servo motors cannot be vibrated incessantly without breaking. The fitness of controllers was the following function of d, the maximum distance traveled: 2^{d^2}. The exponential nature of the function magnified the selective advantage of small increases in the distance traveled. Because HyperNEAT greatly outperforms FT-NEAT on this problem [3], we compare HybrID to only HyperNEAT.

HybrID should excel when a problem is mostly regular but has some irregularities. Adding faulty joints to the quadruped provides such a problem. In addition to experiments with no faulty joints, we also conducted tests where the proportional controller of 1, 4, 8 or all 12 joints had an error such that if the ANN specified a desired target angle of A, the actual desired angle fed to the proportional controller was A + E, where E is an error value in degrees in the range [-2.5, 2.5]. The value E was set once at the beginning of each run for each faulty joint and was constant throughout the run. This is analogous to expected variation in joint function due to manufacturer error.

The results show that HybrID yielded an improvement over HyperNEAT on all versions of this problem (Fig. 3, $p < 0.001$, Wilcoxon rank-sum test). The improvement of HybrID over HyperNEAT was 6, 10, 30, 40, and 38 percent, respectively, for treatments with 0, 1, 4, 8, and 12 faulty joints. The performance boost from HybrID

tended to increase with more faulty joints, and thus roughly correlated with the irregularity of the problem. Interestingly, HybrID with 1 faulty joint actually outperformed HyperNEAT with 0 faulty joints ($p < 0.05$, Wilcoxon rank-sum test). The results suggest that HybrID can increase performance when it is allowed to fine tune the regularities produced by HyperNEAT.

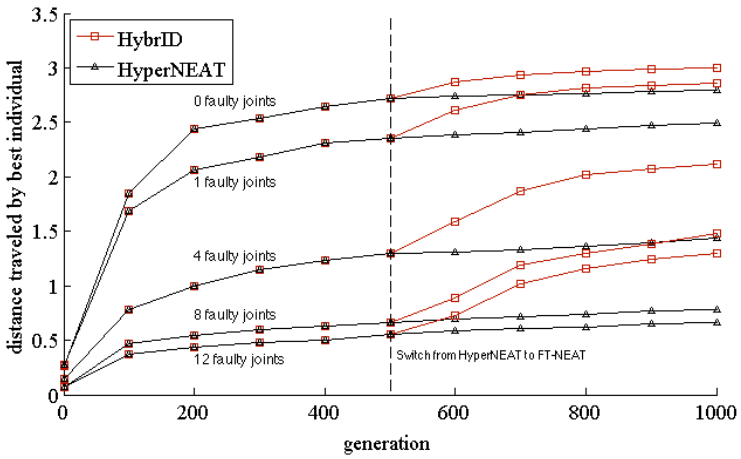

Fig. 3. HybrID outperforms HyperNEAT on all versions of the simulated quadruped controller problem. The increase generally correlates with the number of faulty joints.

4 Discussion and Conclusion

We presented only one of many possible HybrID implementations. The HybrID in this paper evolves first with an indirect encoding then switches to a direct encoding, and could be called a *switch-HybrID*. Another candidate HybrID implementation would have the indirect encoding produce a set of values (e.g., link weights) and the direct encoding evolve a set of offsets that modify the individual values. This *offset-Hybrid* would allow exceptions to be made while the indirect encoding is still evolving. HybrIDs can also be made with other indirect encodings, and in domains besides neuroevolution. Additionally, instead of occurring at a predefined time, the switch from indirect to direct encodings, or the addition of offsets, could occur automatically after fitness has stagnated. Future investigations are required to test the efficacy of different HybrID implementations.

This work also suggests that it is difficult for evolutionary encodings to simultaneously discover the regularity of problems and make exceptions to account for problem idiosyncrasies. At a minimum, HyperNEAT exhibits this deficiency in its present form. Theoretically, HyperNEAT should be able to make any exception required, but in practice it frequently does not. We predict that similar problems exist with other generative encodings, although additional research is required to test our prediction. It is an open challenge for the field to improve current generative encodings to enable

the encoding of both regularities and exceptions to those regularities. If such developments are made to generative encodings, it might obviate the need for HybrID algorithms, but until that time HybrID remains an effective enhancement of generative encodings.

Many real world problems have regularities but also require exceptions to be made. It is important for evolutionary algorithms to both exploit such regularity in problems and account for their irregularities. We have shown that HybrID, a combination of indirect and direct encodings, accomplishes this goal by first discovering the regularity inherent in a problem and then accounting for its irregularities. We validated the algorithm on two simpler test problems and on a more challenging, real-world problem. HybrID frequently outperformed HyperNEAT, sometimes by as much as 40%. While further research is needed to see how HybrID works with other pairs of indirect and direct encodings, alternate HybrID implementations, and on additional problems, these preliminary results suggest that HybrID is an effective algorithm for evolving solutions to complex problems.

Acknowledgments. NSF Grants CCF-0750787, CNS-0751155, and CCF-0820220, U.S. Army Grant W911NF-08-1-0495, and a Quality Fund Grant from MSU. Thanks to helpful comments from Ken Stanley, Heather Goldsby and Dave Knoester.

References

[1] G. S. Hornby, H. Lipson, and J. B. Pollack. Generative representations for the automated design of modular physical robots. *IEEE Transactions on Robotics and Automation*, 19(4):703–719, 2003.

[2] K. O. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.

[3] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock. Evolving coordinated quadruped gaits with the hyperneat generative encoding. In: *IEEE Congress on Evolutionary Computing (CEC)*, pages 2674-2771, Trondheim, Norway, 2009.

[4] J. Clune, C. Ofria, and R. T. Pennock. How a generative encoding fares as problem-regularity decreases. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature (PPSN)*, pages 358-357, Berlin, Heidelberg, 2008.

[5] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3:151–183, 1994.

[6] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[7] J. Grimbleby. Automatic analogue circuit synthesis using genetic algorithms. *IEE Proceedings - Circuits, Devices and Systems*, 147(6):319–323, Dec 2000.

[8] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 15(2), 2009.

[9] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, 2007.

[10] J. Clune, C. Ofria and R. T. Pennock. The sensitivity of hyperneat to different geometric representations of a problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 675-682, Montreal, Canada, 2009.

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