Exploring the Evolution of Internal Control Structure using Digital Enzymes

Chad M. Byers, Betty H.C. Cheng, and Philip K. McKinley
BEACON Center for the Study of Evolution in Action
Department of Computer Science and Engineering
Michigan State University
{byerscha, chengb, mckinle3}@msu.edu

ABSTRACT

The Digital Enzyme [2] model of control is based on the bottom-up, reactive process of signal transduction found in cells. An earlier study applied a specific instance of the this model to the foraging problem. Here, we extend the system and use it to explore a fundamental question in both biology and evolutionary computation, namely, whether environmental complexity is a driving factor for an organism's internal control structure. To address this question, we extended the original system to allow the open-ended evolution of the unique programs, instructions, and threads within each controller. With the extended model, we were able to evolve successful foraging strategies that nearly doubled the performance of strategies found in the earlier work. In response to increasing environmental complexity, we discovered a high degree of variation for the number of programs, threads, and instructions that produced successful strategies. These results highlight the importance of evolutionary search techniques that enable the open-ended evolution of key internal control components.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods, and Search*

General Terms

Experimentation

Keywords

Artificial life, digital evolution, digital enzyme, digital signal transduction, environmental complexity, evolution, foraging

1. INTRODUCTION

As the basic unit of life, the cell performs an ongoing process of sensing information from the environment in the form of energy and molecules, harnessing these molecules to drive internal reactions, and ultimately, producing a response. This biological control process strongly parallels many human-designed embedded system controllers, where hardware sensors collect information from the environment, perform internal computations, and translate the results into specific activations of actuators. Sensing information from the environment and activating actuators are relatively straightforward tasks in the design of embedded controllers. However, the internal decision process used to translate environmental information into meaningful, robust behaviors is often a *black box* design question, as shown at the top of Figure 1.

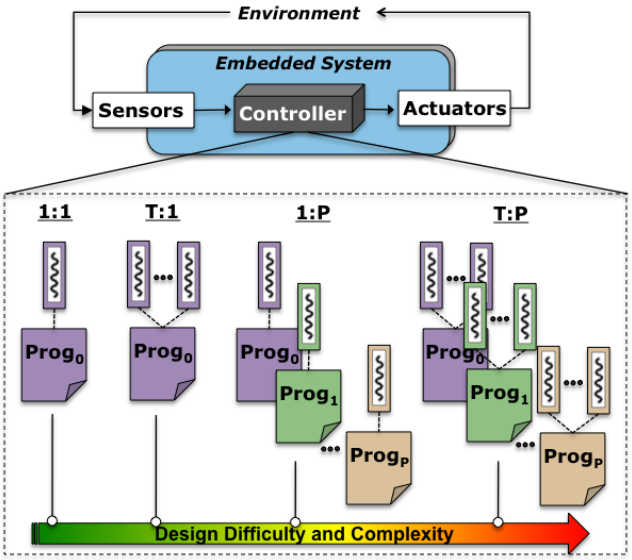

Figure 1: [Top]: A typical embedded system containing sensors and actuators interacting with the environment. [Bottom]: A broad range of controller designs are possible to produce the desired system behavior, ranging from single-program-single-instance designs (1:1) to multi-program-multi-thread designs (T:P).

The design used to implement a controller's decision logic can take on a vast number of possibilities with respect to the number of parallel programs (P) and the number of threads (T) executing each parallel program. How should the decision logic be implemented within the controller with respect to these two variables? Should control be managed by one executing thread of one program (a **1:1** design), many executing threads of one program (**T:1**), one executing thread of many different programs (**1:P**), or many executing threads of many different programs (**T:P**)? Of course, these designs are only key transition points along an entire spectrum towards more distributed control, as shown at the bottom of Figure 1. Advantages and disadvantages in terms of human design efforts and performance tradeoffs exist along this spectrum. For example, the 1:1 controller design is pervasive because a developer is not burdened by synchronization nor the overhead required to support parallel execution, such as memory and threading libraries.

GECCO'12 Companion, July 7–11, 2012, Philadelphia, PA, USA.
ACM 978-1-4503-1178-6/12/07.

On the other hand, parallelism and multithreading might greatly enhance functionality and improve performance. However, as controller complexity moves farther to the right along this spectrum, it becomes more difficult for developers to conceptualize the decision logic while also providing beneficial qualities such as cooperation, robustness, fault tolerancy, and efficiency, especially when applying a top-down design approach.

An alternative approach involves uses a bottom-up, population-based search technique, evolutionary computation. By allowing random mutation to make changes to each of these design features, the developer can study the internal control structures *naturally* selected for during evolution as well as their relationship to environmental complexity. Biological evolution is theorized to have already provided us with *one* example of a natural progression from a 1:1 RNA-dominated world to the complex, DNA-dominated T:P lifeforms inhabiting the diverse environments of the world today [4]. Perhaps evolutionary computation will also naturally select for a distributed control design to drive organisms into more challenging, complex environments or to maintain the integrity of the solution within these environments.

An earlier study [2] introduced the digital enzyme (DE) model of controller design, which is based on the biological process of signal transduction within a cell. Within each controller, stimuli from the environment are encoded into bitstrings that can be loaded, manipulated, and outputted to actuators for mapping to specific responses. An evolved controller's decision logic is encoded in assembly-like programs, referred to as genes, each being executed in parallel by threads, referred to as digital enzymes. In the model described in [2], the number of genes and digital enzymes was statically defined, allowing only instructions to mutate during evolution. To address the fundamental question of whether environmental complexity influences internal control structure, we modified the original design to allow (1) the number of genes, (2) their instructions, and (3) the number of digital enzymes executing each gene, to evolve freely.

To experimentally control an increase in environmental complexity and to provide a direct comparison to previous results, we applied this system to a unique version of the central-place foraging problem. Each evolved controller was evaluated in a homogeneous colony on its ability to find and return food from a randomly placed clump. As an individual strategy became more successful, the distance that food was placed from home's perimeter during evaluation was increased. We imposed a maximum bound on the food placement distance within each evolutionary run, allowing us to analyze the bound's effect on the average number of genes, enzymes, and instructions selected for.

With respect to the evolution of parallel programs, a variety of approaches exist in the area of genetic programming [1] including Cartesian Genetic Programming [5, 8], Parallel Distributed Genetic Programming [6], Genetic Parallel Programming [3], and Concurrent Genetic Programming [7]. However, each of these approaches contain one of the following drawbacks: (1) the maximum size of the genetic program's grid, graph, or tree must be predefined, (2) a crossover operator imposes a high rate of disruption and genetic bloat, (3) instructions are coarse-grained requiring each actuator action to be encoded as an instruction, or (4) their application does not include evolving reactive controllers. In contrast, the digital enzyme model does not impose restrictions on the size of programs within a controller's genome and each controller is asexual to avoid the disruptive effect of crossover, often found in sexually reproducing systems. Moreover, the previously mentioned approaches do not explore the effect of environmental complexity on the number of parallel programs executed within an evolved solution.

The results of our experiments demonstrate that open-ended evolution enabled the discovery of more effective solutions, nearly doubling the performance of previous foraging solutions [2]. In response to increasing environmental complexity, we found a high degree of variation in the number of programs, threads, and instructions necessary for successful strategies. For researchers in evolutionary computation, these results reveal that environmental complexity does not impose *requirements* on the types of evolutionary search methods that are used to find solutions. In this study, evolutionary search techniques employing a single program, single thread approach would find strategies equally as successful as techniques using multiple programs and threads. However, the approaches limited to a single program and thread would find only a *subset* of the evolved successful strategies. The discovery of successful strategies within regions of the search space characterized by parallel and distributed control emphasizes how techniques, enabling open-ended evolution, can successfully explore larger regions of the search space.

2. ACKNOWLEDGMENTS

This work has been supported in part by NSF grants CCF-0541131, CNS-0751155, IIP-0700329, CCF-0820220, DBI-0939454, CNS-0854931, CNS-1059373, CNS-0915855 Army Research Office grant W911NF-08-1-0495, Ford Motor Company, and a Quality Fund Program grant from Michigan State University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, U.S. Army, Ford, or other research sponsors.

3. REFERENCES

[1] F. H. Bennett III. Automatic creation of an efficient multi-agent architecture using genetic programming with architecture-altering operations. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 30–38. MIT Press, 1996.

[2] C. M. Byers, B. H. Cheng, and P. K. McKinley. Digital enzymes: agents of reaction inside robotic controllers for the foraging problem. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 243–250, New York, NY, USA, 2011. ACM.

[3] S. M. Cheang, K. S. Leung, and K. H. Lee. Genetic parallel programming: Design and implementation. *Evolutionary Computation*, 14:129–156, June 2006.

[4] W. Gilbert. Origin of life: The RNA world. *Nature*, 319(6055):618, Feb. 1986.

[5] J. Miller. *Cartesian Genetic Programming*. Natural Computing. Springer, 2011.

[6] R. Poli. Evolution of graph-like programs with parallel distributed genetic programming. In *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 346–353, East Lansing, Michigan, 1997. Morgan Kaufmann.

[7] A. Trenaman. Concurrent genetic programming, tartarus and dancing agents. In *Proceedings of the 2nd European Workshop on Genetic Programming*, pages 270–282, London, UK, 1999. Springer-Verlag.

[8] J. A. Walker, K. Völk, S. L. Smith, and J. F. Miller. Parallel evolution using multi-chromosome cartesian genetic programming. *Genetic Programming and Evolvable Machines*, 10:417–445, December 2009.