GEF: A Self-Programming Robot Using Grammatical Evolution

Charles Peabody and Jennifer Seitzer
Department of Mathematics and Computer Science
Rollins College, Winter Park, FL 32789
{cpeabody, jseitzer}@rollins.edu
http://myweb.rollins.edu/JSEITZER

Abstract

Grammatical Evolution (GE) is that area of genetic algorithms that evolves computer programs in high-level languages possessing a BNF grammar. In this work, we present GEF ("Grammatical Evolution for the Finch"), a system that employs grammatical evolution to create a Finch robot controller program in Java. The system uses both the traditional GE model as well as employing extensions and augmentations that push the boundaries of goal-oriented contexts in which robots typically act including a meta-level handler that fosters a level of self-awareness in the robot.

To handle contingencies, the GEF system has been endowed with the ability to perform meta-level jumps. When confronted with unplanned events and dynamic changes in the environment, our robot will automatically transition to pursue another goal, changing fitness functions, and generate and invoke operating system level scripting to facilitate the change. The robot houses a raspberry pi controller that is capable of executing one (evolved) program while wirelessly receiving another over an asynchronous client. This work is part of an overall project that involves planning for contingencies. In this poster, we present the development framework and system architecture of GEF, including the newly discovered meta-level handler, as well as some other system successes, failures, and insights.

Introduction

Machine learning is an area of artificial intelligence in which computer systems are built to extract patterns and knowledge from data for the purpose of solving complex tasks [Russel2010]. One method of machine learning is the use of Genetic Algorithms (GA) inspired by mechanisms of biological evolution [Booker1989]. The work presented here uses a derivative of GAs known as Grammatical Evolution (GE) [Ryan1998] of which the computer system GEF ("Grammatical Evolution for the Finch") was built to automatically create an optimal robot controller-program to direct a Finch robot to solve simple navigation problems.

Grammatical Evolution

Genetic algorithms (GA) is the basis for evolutionary computation which iteratively evolves solutions to problems using a continuous iterative algorithm of *generate–measure–select* of evolving solutions called *chromosomes* [Holland 1995].

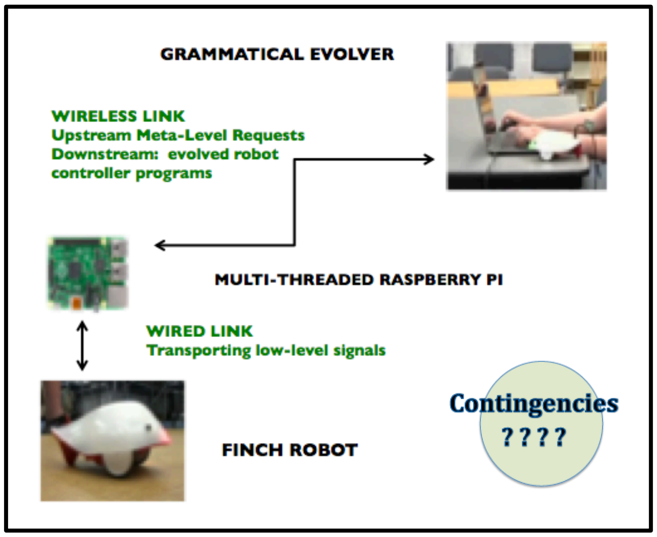
GRAMMATICAL EVOLVER
WIRELESS LINK
Upstream Meta-Level Requests
Downstream: evolved robot controller programs
MULTI-THREADED RASPBERRY PI
WIRED LINK
Transporting low-level signals
FINCH ROBOT
Contingencies
????

Fig.1 System GEF Components and Behavior [Seitzer2014]

Genetic programming (GP) uses the techniques established in GA to derive solutions in the form of LISP programs – that is, it evolves computer programs in the language of LISP [Koza 1992]. Grammatical evolution (GE) extends the work of GP by evolving computer programs in any imperative computer language that has an associated Bakus-Naur Form (BNF) grammar. In GE, the high-level programming language (e.g., Java) is represented in a data structure with a format spawned from the BNF representation and exploits the multiple options inherent in grammar rules to exhaustively create the many different possibilities of programming statements.

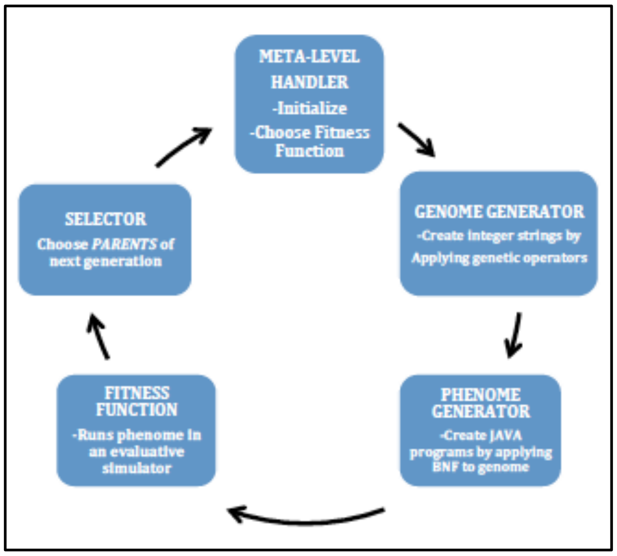
META-LEVEL HANDLER
-Initialize
-Choose Fitness Function
GENOME GENERATOR
-Create integer strings by Applying genetic operators
PHENOME GENERATOR
-Create JAVA programs by applying BNF to genome
FITNESS FUNCTION
-Runs phenome in an evaluative simulator
SELECTOR
Choose PARENTS of next generation

Figure 2: GEF System Architecture and Algorithm

An autonomous robot uses on-board controllers to perceive, decide, and act on their environment [Russell 2010]. There is a growing demand for autonomous robots to act intelligently in dynamic environments. In this work, we are applying grammatical evolution to create optimal robot-controlling programs to be executed by the robot controller of the Finch robot.

Novel Aspects of System GEF

The long-term goal of this research is to produce *real-time* evolution of an optimal robot controller program. By *real-time,* we mean evolution of new versions of a controller program during execution of a former version. To date, we have been successful in partially achieving this goal by integrating into system GEF a *meta-level handler* (as shown above in the top module of Figure 2). At any given point, the robot is executing the most recent, most appropriate, controlling program that had been evolved using the traditional framework embodied in the other four modules of Figure 2: genome generator, phenome generator, fitness function, and selector [Ryan1998].

As shown in Figure 1, GEF has three loci of computation: (1) the laptop housing the grammatical evolution engine and a server to push evolved programs, (2) the raspberry PI housing the scripts to receive "alert messages" from the Finch as well as processes executing the current controller program and serving in the role as a client to the program-transporter on the laptop, and (3) the Finch robot, itself, following the commands mandated in the current controller program. All three of these entities possess their own operating systems with corresponding socket call APIs and scripting languages that enable client-server interactions involving the transfer of meta-programs (i.e., scripts) as well as object-level programs (i.e., Finch controllers).

As implied in Figure 1, GEF is continually transferring, decoding, and responding to messages (in the form of Finch controller programs). Additionally, each evolved program, itself, is intentionally endowed with meta-level scaffolding including a software-level program counter that enables the Finch to create an *alert message* such as: "Alert: send new program; completed instruction 8." The 'alert' messages are directly mapped to a vocabulary of contingencies among some of which include *unavoidable obstacle* (e.g., a wall), or *internal meta event* (e.g., right wheel broke off or battery low).

Incremental Testing and Future Plans

Our first meta-level jumping experiment evolved a program that enabled the Finch to successfully navigate the proper exit of a room through a specified door. We then intentionally closed that door at which point the robot had to (a) realize there was a problem, (b) alert the meta-script with the current instruction number, and (c) wait for another program to be wirelessly downloaded. Among other meta-triggering events, we elaborated the system to handle internal-meta-problems (i.e., self awareness of blocked right-wheel / left-wheel). The next of our future plans is to "bank" effective (previously evolved) programs as well as engage a real-time evolver to handle a broader repertoire of contingencies including environmental and architectural anomalies some of which include holes in the floor and faulty sensors.

References

Booker, L. B., Goldberg, D. E., and Holland, J. H. 1989. Classifier systems and genetic algorithms. *Artificial Intelligence* 40(1/3), 235-282.

Holland, J. H.,1995. *Hidden Order: How Adaptation Builds Complexity*. Reading, Mass.:Addison-Wesley.

Koza, J. R.,1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, Mass.: MIT Press.

O'Neill M., Ryan C. Under the Hood of Grammatical Evolution. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (eds.). *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, July 13-17, 1999, Orlando, Florida USA*. San Francisco, CA: Morgan Kaufmann.

Russell, S., Norvig, P., and Davis, E. 2010. *Intelligent Agents Artificial Intelligence: A Modern Approach 3rd Edition*. Upper Saddle River, NJ:Pearson-Prentice Hall.

Ryan C., Collins J.J., O'Neill M. 1998. Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Proceedings of the First European Workshop on Genetic Programming*. London, United Kingdom: Lecture Notes in Computer Science 1391.