Incorporating Expert Knowledge in Object-Oriented Genetic Programming

Michael Richard Medland
Dept. of Computer Science
Brock University
St. Catharines, ON, Canada
mm08sj@brocku.ca

Kyle Robert Harrison
Dept. of Computer Science
Brock University
St. Catharines, ON, Canada
kh08uh@brocku.ca

Beatrice Ombuki-Berman
Dept. of Computer Science
Brock University
St. Catharines, ON, Canada
bombuki@brocku.ca

ABSTRACT

Genetic programming (GP) has proven to be successful at generating programs which solve a wide variety of problems. Object-oriented GP (OOGP) extends traditional GP by allowing the simultaneous evolution of multiple program trees, and thus multiple functions. OOGP has been shown to be capable of evolving more complex structures than traditional GP. However, OOGP does not facilitate the incorporation of expert knowledge within the resulting evolved type. This paper proposes an alternative OOGP methodology which does incorporate expert knowledge by the use of a user-supplied partially-implemented type definition, i.e. an abstract class.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program Synthesis*

Keywords

Expert Knowledge; Object-Oriented Genetic Programming

1. INTRODUCTION

Genetic programming (GP) [4] is an artificial intelligence paradigm which uses the concept of Darwinian evolution to automatically synthesize computer programs. While GP provides the means to generate simple solutions quite easily, using GP to evolve sophisticated programs which consist of many functions is challenging. Understanding the output and, therefore, the behaviors of evolved programs is also a significant challenge.

GP has been shown to produce fit programs in a wide variety of different applications [5]. Likewise, there have been a variety of methodologies introduced to extend and enhance the capabilities of GP [1, 3, 5].

Linear GP [3], proposed as an extension to traditional GP, took a more imperative approach to program representation by using array structures for chromosomes. While linear GP effectively modeled the imperative programming paradigm, the concept of entities with multiple behaviors was not facilitated.

Object-oriented GP (OOGP)[1] introduced a multi-tree representation allowing multiple functions, corresponding to the constituent behaviors of an object, to be simultaneously evolved. While OOGP facilitates the evolution of objects using GP, the behavior of the evolved objects is entirely defined by the output of the GP system. Partial solutions, known *a priori*, cannot be easily incorporated within the resulting program using OOGP.

This paper proposes a novel GP paradigm which combines the imperative style of linear GP with the object-oriented approach of OOGP while allowing the incorporation of expert knowledge in the resulting programs. The proposed GP system, LinkableGP, makes use of a partially-implemented type definition, i.e., an abstract class, to allow expert knowledge to be embedded within the evolved program.

2. FACILITATION OF EXPERT KNOWLEDGE

To facilitate expert knowledge in GP, a novel object-oriented linear GP system, LinkableGP, is proposed. LinkableGP makes use of 1) an object-oriented methodology to construct class-based programs and 2) linear GP to construct imperatively-defined methods within the class. Expert knowledge is incorporated by supplying a partially-implemented class, i.e., an abstract class, to the system. This abstract class allows the user to incorporate domain knowledge of the problem by providing the implementation of methods where the desired functionality is known *a priori*. The unimplemented methods from the abstract class are evolved using LinkableGP's evolutionary strategies. Thus, each individual in the LinkableGP system is a stand-alone subclass derived from the supplied abstract class.

2.1 Structure and Representation

The structure of individuals within a population in LinkableGP are inspired by both linear GP and OOGP representations. Individuals are represented by a collection of chromosomes, each defining a derived implementation of an abstract (i.e., unimplemented) method from the supplied abstract class. Each chromosome contains an array of integers which, during the genotype to phenotype conversion process, translates to a code sequence that constructs a single method in the resulting phenotype.

The genotype to phenotype conversion process consists of using each chromosome to select a sequence of functions

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
ACM 978-1-4503-2881-4/14/07.
http://dx.doi.org/10.1145/2598394.2598494.

corresponding to each method implementation. Selection operations are performed by enumerating the available candidates and using the respective chromosome value in a modulo fashion. In order to build each method, a language context (i.e., a set of functions, constants, and constant generators) must be defined. Construction of the method starts with a variable bag containing, initially, the parameters of the method being implemented. The first value from the chromosome's integer array is used to select a satisfiable[1] function from the language context. Then, successive chromosome values are used to satisfy each argument of the function by selecting a variable with the correct type from those available in the variable bag, constants, and constant generators. Finally, if the selected function has a non-void return type, a further chromosome value is used to select either a mutable variable from the variable bag or a new variable (which is then added to the variable bag). This process continues until all chromosome values are exhausted. Chromosomes are extended as necessary to ensure the completion of each function call.

2.2 Operations

LinkableGP implements a typical, generational genetic algorithm that employs crossover, mutation, elitism, and selection operations. **Crossover** is a two phased operation. The first phase, referred to as mating, selects two parents using a standard tournament selection operator and constructs a bit-mask to determine the chromosomes which are to be inherited from each parent. Each bit in the randomly generated bit-mask acts a parent discriminator, i.e., the chromosome at index i is selected from parent 1 or parent 2 based upon whether the bit at index i is a 0 or 1.

The second phase, referred to as mixing, occurs on each chromosome of the child with probability ρ. During the mixing phase, a chromosome in the child is replaced by the offspring resulting from a one-point crossover operation between the parents. A random cut-point is selected within the shortest parent chromosome. The mixing phase occurs in one of two ways, with equal probability: 1) the genetic material up to the cut-point is taken from parent 1 while the genetic material after the cut-point is taken from parent 2, or, 2) the genetic material up to the cut-point is taken from parent 2 while the genetic material after the cut-point is taken from parent 1.

Mutation in LinkableGP is done either by extending the length of a chromosome or by randomly changing values within a chromosome. Extension mutation and random change mutation operations are performed with a probability of m_1 and m_2, respectively, on each chromosome of a child produced during crossover.

2.3 Example

A generalized algorithm, shown in Algorithm 1, is proposed to construct a graph model based on *a priori* knowledge of the expected behavior for existing graph models. In this example, what is knownn is that the graph will have some initial state (*initializeGraph*) but that state is unclear. After the graph is initialized, it is known that for some given number of iterations, t, a new vertex, v, will be added to the graph. Furthermore, n existing vertices, denoted collectively as S, will be selected. Finally, each vertex in S will be connected to v. However, it is not known exactly how the methods *initializeGraph*, *selectNumber*, *selectVertices*, and *createEdge* should work. Thus, they are evolved by LinkableGP where each are represented via a chromosome of an individual. The conversion between genotype to phenotype here results in a fully implemented graph model class.

[1]All of its arguments can be satisfied from the variable bag, the collection of constants, or constant generators

Algorithm 1: Generalized Graph Model

```
Data: t number of time steps
Result: A graph
g ← initializeGraph();
for i ∈ 1 : t do
    v ← g.AddVertex(), n ← selectNumber();
    S ← selectVertices(g, n);
    for s ∈ S do
        createEdge(g, v, s);
    end
end
return g;
```

3. CONCLUSION

This paper proposed a novel genetic programming (GP) paradigm which facilitates the incorporation of expert knowledge within the evolved program structure. The proposed GP system, LinkableGP, was inspired by both linear GP and object-oriented GP (OOGP) methodologies. However, the LinkableGP system combined the benefits of each approach by allowing the simultaneous evolution of multiple imperative-styled methods. LinkableGP uses a representation whereby an individual is comprised of multiple chromosomes, each of which directly correspond to a method which is to be evolved. Furthermore, LinkableGP facilitates expert knowledge through partially-implemented types, allowing the user to embed portions of the solution known *a priori* within the resulting individuals.

4. REFERENCES

[1] R. Abbott. Object-oriented genetic programming, an initial implementation. In *International Conference on Machine Learning: Models, Technologies and Applications*, pages 26–30, 2003.

[2] A. Bailey, M. Ventresca, and B. Ombuki-Berman. Automatic generation of graph models for complex networks by genetic programming. In *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference*, GECCO '12, pages 711–718, New York, NY, USA, 2012. ACM.

[3] M. Brameier and W. Banzhaf. Explicit control of diversity and effective variation distance in linear genetic programming. In J. Foster, E. Lutton, J. Miller, C. Ryan, and A. Tettamanzi, editors, *Genetic Programming*, volume 2278 of *Lecture Notes in Computer Science*, pages 37–49. Springer Berlin Heidelberg, 2002.

[4] J. R. Koza. *Genetic Programming: vol. 1, On the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[5] R. Poli, W. W. B. Langdon, N. F. McPhee, and J. R. Koza. *A field guide to genetic programming*. Lulu. com, 2008.