Charles University in Prague

Faculty of Mathematics and Physics

BACHELOR THESIS

Petr Mánek

Genetic programming in Swift for human-competitive evolution

Department of Software and Computer Science Education

Supervisor of the bachelor thesis:	RNDr. František Mráz, CSc.
Study programme:	Computer Science
Study branch:	General Computer Science

Prague 2016

Title: Genetic programming in Swift for human-competitive evolution

Author: Petr Mánek

Department: Department of Software and Computer Science Education

Supervisor: RNDr. František Mráz, CSc., Department of Software and Computer Science Education

Abstract: Imitating the process of natural selection, evolutionary algorithms have shown to be efficient search techniques for optimization and machine learning in poorly understood and irregular spaces. In this thesis, we implement a library containing essential implementation of such algorithms in recently unveiled programming language Swift. The result is a lightweight framework compatible with Linux-based computing clusters as well as mobile devices. Such wide range of supported platforms allows for successful application even in situations, where signals from various sensors have to be acquired and processed independently of other devices. In addition, thanks to Swift's minimalistic and functional syntax, the implementation of bundled algorithms and their sample usage clearly demonstrates fundamentals of genetic programming, making the work usable in teaching and quick prototyping of evolutionary algorithms.

Keywords: genetic programming artificial evolution

Dedication.

Contents

Introduction

Evolutionary Algorithms

TODO

Genetic Programming

TODO

The Swift Language

TODO

Practical Application

TODO

Structure of This Document

TODO

1. Object-oriented Design

In this chapter, the high-level design of individual components of the library is described.

Anděl [2007] **TODO**

1.1 Random Genereration

Randomness plays a crucial role in evolutionary algorithms. Since the properties of pseudo-random generators impact the quality of produced solutions significantly, the library gives users full control over the algorithm, which is used to produce random sequences. In object design, this is achieved by simple abstraction.

The functionality of a random number generator is facilitated by *an entropy generator* object. In runtime, only a single instance of such object is created. This instance is then passed on to other components of the library, which require its capabilities. These components access the entropy generator by reference. Users are responsible for instantiating this object, and can thus specify a seed for the generator or choose an algorithm particularly suitable for their application.

For the sake of minimality, entropy generators are only required to produce positive floating-point decimals from the $[0; 1]$ interval. In spite of that, they can be used to generate random values of various types. This mechanism provided that the generated decimals can be mapped onto the type while maintaining uniform distribution of generated values. This is further discussed in section 1.1.1.

1.1.1 Data Structures

Every individual in a generation is repesented by a separate instance of a class. The primary responsibility of such object is to store genetic information, which defines the individual. This information does not need to be held in a homogeneous data structure. In fact, it can be stored in any type suitable for the application. The only requirement on such type is that it can be generated randomly.

TODO

1.1.2 Randomizable Interface

TODO

1.1.3 Discrete Interface

TODO

1.2 Genetic Operators

TODO

1.2.1 Operator Life Cycle

TODO

1.2.2 Custom Interfaces

TODO

1.3 Selections

TODO

1.4 Algorithms

TODO

2. Library Implementation

TODO

2.1 Data Structures

In this section, the data structures capable of holding genetic information are described. To help first-time users, the library includes implementation of several types, commonly used in genetic programming. It does however also include a set of requirements in the form of Swift interfaces, which allow advanced users to implement their own dedicated data structures and customize their behavior.

Users are thus free to choose, whether they wish to utilize already implemented data structures or create their own. Moreover, data structures distributed along with the library also support nesting. Users can combine them in whatever way in order to form more complex structures.

2.1.1 Common Types

TODO

2.1.2 Tree Structures

TODO

2.1.3 Program Interpretation

TODO

2.2 Genetic Operators

TODO

2.2.1 Reproduction

TODO

2.2.2 Mutation

TODO

2.2.3 Crossover

TODO

2.3 Selections

TODO

2.3.1 Roulette Selection

TODO

2.3.2 Rank Selection

TODO

2.3.3 Tournament Selection

TODO

2.3.4 Extensions

TODO

2.3.5 Optimizations

TODO

2.4 Algorithms

TODO

2.5 Event-driven Approach

TODO

2.6 Extensions

TODO

3. Usage Demonstration

TODO

3.1 Trivial Examples

TODO

3.2 Self-driving Car Simulation

TODO

3.3 QWOP Player

TODO

Conclusion

Deployment

TODO

Teaching

TODO

Applications

TODO

Bibliography

J. Anděl. *Základy matematické statistiky.* Druhé opravené vydání. Matfyzpress, Praha, 2007. ISBN 80-7378-001-1.

List of Figures

List of Tables

List of Abbreviations

Attachments