CodeMonkey; a GUI Driven Platform for Swift Synthesis of Evolutionary Algorithms in Java

Reza Etemadi, Nawwaf Kharma, and Peter Grogono

Electrical and Computer Engineering Department, Concordia University, Montreal (QC), Canada H3G 1M8
{r_etemad@encs,kharma@ece,grogono@cse}.concordia.ca

Abstract. CodeMonkey is a GUI driven software development platform that allows non-experts and experts alike to turn an evolutionary algorithm design into a working Java program, with a minimal amount of manual code entry. This paper describes the concepts behind CodeMonkey, its internal architecture and manner of use. It concludes with a simple application that exhibits its utilization for multi-dimensional function optimization. CodeMonkey is provided free of charge, for non-commercial users, as a plug-in for the Eclipse platform.

Keywords: Evolutionary Algorithm, Java Language, Eclipse Platform, GUI Application

1 Introduction

CodeMonkey is a GUI-driven software platform that allows non-expert users to generate an executable Java implementation of a custom-designed Evolutionary Algorithm, meant for a particular optimization or design application.

1.1 Review

There are many types of Evolutionary Algorithms (EA) including Genetic Algorithms (GA), Genetic Programming (GP), Evolutionary Strategies (ES) and Evolutionary Programming (EP). Software platforms differ in: scope (some are more generic while others are specialized to certain domains); performance and scalability (based on the language or platform they employ); usability (whether it is just a library, framework, application or platform). A few widely-used EA development aides follow.

GEATbx [1] (Genetic and Evolutionary Algorithm Toolbox) is a Matlab tool, and one of the few packages that cover the four main flavors of EA. It allows users to define homogeneous genotypical representations (i.e., lists of one type of variable); it has limited support for heterogeneous genotypes (with different types in the same structure) [2]. It offers a GUI for novice users. It has a limited selection of genetic variation operations. It is a proprietary package and is not extendable by third parties.

Evolving Objects (EO) [3] is a template-based C++ open-source framework for writing stochastic optimization programs. It supports homogeneous genotypical representations (but not integers). It does not offer readily usable heterogeneous genotypes. Many selection mechanisms are provided. It does not allow GUI based customization. It is open-source and free to use under GPLv2. It is extendable by third-parties.

DREAM (Distributed Resource Evolutionary Algorithms Machine) [4][5] software framework (called Java Evolving Object) defines many genotypes together with matching variation operations. It does not provide heterogeneous genotypes. It has a GUI for non-programmers for both I/O interaction and problem definition. It is open-source and is offered under a GPL license. The platform has a distributed architecture, and the framework has an API that facilitates distributed implementations.

Watchmaker Framework for Evolutionary Computation [6] is a framework for implementing evolutionary algorithms in Java. It is used in the Apache Mahout Project, and some specialized frameworks, such as GEP4J. It does not provide predefined genotypes. There is a GUI for monitoring progress, but it does not offer a GUI for non-experts to generate code. There are a limited number of variation operators. It is open-source and available under Apache software license, and it is easily extendable.

Table 1. Camparative Summary of Common EA Platforms

Name	GEATbx	Evolving Objects	DREAM	Watchmaker	JGAP
Programming Language	Matlab	C++	EASEA /Java	Java	Java
Type	Library	Framework	Platform	Framework	Framework
Homogeneous Genotypes	Real, Integer, Binary, Permutation	Real, Binary, Permutation	Real, Integer, Binary, Tree	None	Real, Integer, Binary, String & more
Heterogeneous Genotypes	None	None	None	None	None
Selection Types	4	9 (more pluggable)	11 (more pluggable)	6 (more pluggable)	4 (more pluggable)
GUI for Novice users	Yes	No	Yes	No	No
Open Source	No	Yes	Yes	Yes	Yes
Extendable	No	Yes	Limited	Yes	Yes
Licencing /Pricing	Multi-tier /Not-free [8]	GPLv2 /Free	GPL /Free	Apache v2.0 /Free	LGPL,Mozilla /Free

JGAP (Java Genetic Algorithms Package) [7] is a GA and GP component provided as a Java framework. It provides basic genetic mechanisms that can be used to apply evolutionary principles to problem solutions. It has many ready-to-use genotypes defined, but does not offer heterogeneous genotypes. It also does not provide a GUI for end-users to generate code; only expert users, who must learn the framework and have prior knowledge of GA or GP, can make use of JGAP. It is, however, an open-source resource, free and extendable under LGPL and Mozilla licenses. Table 1 provides a brief comparison of the EA platforms outlined in this section.

The comparison table demonstrates that only GEATbx and DREAM offer a customization GUI for non-expert users. None of them attempts to provide support for heterogeneous genotypes. In contrast, we provide an easy-to-learn and use GUI-driven platform for the generation of EAs of different flavors and for many target application. It supports both homo- and heterogeneous genotypes with appropriately defined variation operators. It also offers a great degree of flexibility in both offspring generation and survivor selection; a full description of its internal design and manners of use follows. The paper concludes with a simple use example.

2 Design & Implementation

CodeMonkey (CM) is a project that takes advantage of modern features of the Java language, such as Generics and Annotation, and combines that with the power and ease of use of the Eclipse platform, to provide both a framework for expert users and an Eclipse plug-in application for non-expert developers of Evolutionary Algorithms.

The framework aspect of CM is not unique, but Eclipse integration and its step by step GUI wizard allow users with little knowledge of programming to generate serious EA programs that are readily executable.

2.1 Concept

This section offers a description of CMs main configuration steps. This is interleaved with explanations of the main conceptual innovations that were employed in order to make CM as generic as it is.

Genotype Representation: CM divides genotype representations into two categories: homogeneous and heterogeneous. A homogeneous genotype is a collection of genes with identical types (e.g., Boolean, integer or real) while a heterogeneous genotype is a collection of homogeneous genes of different types. CM supports basic homogeneous genotypes and allows easy definition of heterogeneous genotypes.

Termination Criteria: CM places the termination conditions which can be combined under three categories: (a) the Goal Achieved category, which means that an acceptable level of fitness has been attained by at least one individual in the population; (b) the Stagnation Reached category, which means that the improvement in fitness over a preset number of generations is too low to justify continuation; (c) the Resources Exhausted category, which means that a preset limit on a computational resource, such as processing time, has been reached or breached.

Parent Selection: there is a wide spectrum of algorithms for parent selection [9], from fully deterministic (such as Truncation) to fully probabilistic (such as Random), which are typically applied to two individuals. Prior to the actual application of any selection method, the raw fitness of an individual can be transformed (e.g., via ranking) into a different value: it is this new value that is used by the selection method. Regarding selection methods, a unique contribution of CM is the way it unifies many different parent selection mechanisms into one window-based selection generic algorithm. This algorithm employs a selection window, which can be as large as the population size or as small as two individuals. Within a window of a certain size, the picking of an individual from the population can occur on a deterministic or probabilistic basis. The result is a parent, which is deposited into the parent pool. For example, binary tournament selection can be viewed as deterministic selection of the fittest individual from a window of size two. While proportional selection can be seen as probabilistic selection from a window that includes the whole population.

Variation Operations: there are crossover and mutation operations suited to different genetic representations. In CM, different operations are defined for different homogeneous and heterogeneous genotypes. Another contribution of CM is that any number of variation operations can be used with different probabilities and in different sequences along one or more paths linking the parent pool to the offspring pool. This allows users of CM to define a GA-like single sequence of variation operations of say crossover followed by mutation, or alternatively define a GP-style tree of variation operations, with crossover working in parallel with mutation to generate offspring.

Survivor Selection: it allows the generation of the next population using individuals from the current population and/or offspring pool. In CM, all selection mechanisms used for parent selection are available to survivor selection. The difference is that we can apply the selection window to any one or both of the current population and the offspring pool. In addition, CM allows the use of elitism and injection.

2.2 The Framework

To explore CodeMonkeys framework, we start by describing its architecture and data model. The class diagram in figure 1 presents the core package of the framework, which includes the main classes described below.

Class Phenotype represents an individual solution. In the core package this is an abstract class that has two other elements, Genotype and Fitness (mapped through generics). In any implementation, a subclass of this class will be needed for representing individuals.

Class Genotype is the class that represents the genetic encoding of Phenotype. It is a Java interface that extends Java Collection interface. For Boolean, integer and real types, the framework provides implementation. Several variation operators are available in the framework for each genotype implementation.

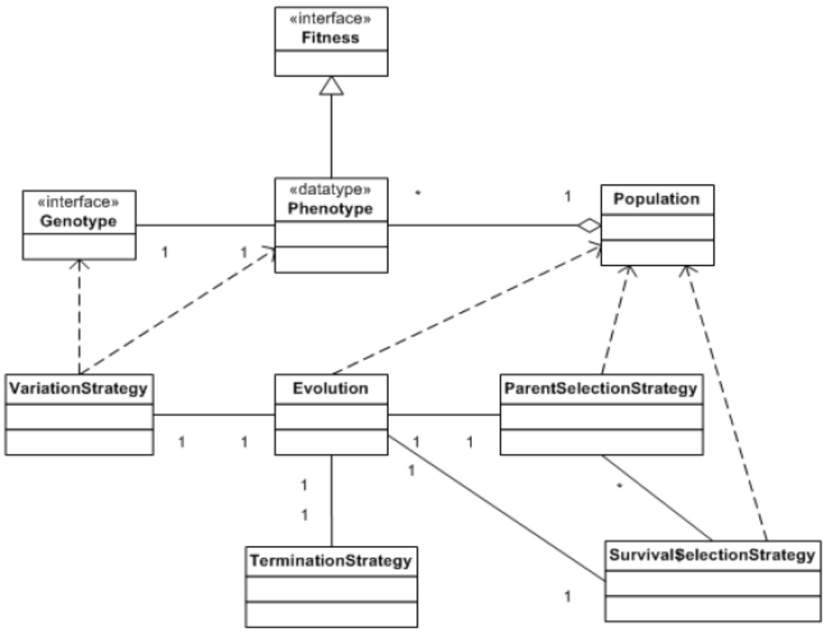

Fig. 1. CodeMonkeys Class Diagram.

These classes are also used for building homogeneous and heterogeneous genotypes in the CM application.

Class Fitness represents the suitability of the phenotype as a solution in comparison to other phenotypes. This is also defined as a Java interface in the framework that extends Java Comparable interface. This interface uses Java generics to accept any subclasses of the Java Number class that implements the Comparable interface.

Class Population represents a collection of Phenotypes. This class (or its subtypes) can be registered to represent the initial population, the parent pool, the offspring pool as well as the next generation. This class has many utility methods for random selection and sorting as well as calculating various statistics of the population (based on fitness and other attributes).

Class TerminationStrategy is an abstract class that represents the termination criteria in the framework. All possible criteria that are used in the CM application are provided in this class. Any concrete subclass can be registered in an implementation. It is invoked periodically to see if the evolutionary process should terminate.

Class ParentSelectionStrategy is the class that realizes the unitary parent selection approach described above. There are three subclasses of this class: one for Truncation selection, one for Proportional selection and a third for Random selection. Any concrete subclass of ParentSelectionStrategy that is registered will be invoked to create the parent pool.

Class VariationStrategy is the abstract class of all variation operations in the framework. Any concrete subclass will invoke the variation method that is implemented in the Genotype with the desired probability and sequence. The class must be registered before it can be invoked to create the offspring pool from the parent pool.

Class SurvivalSelectionStrategy is the abstract class for the survivor selection process in the framework. It internally relies on the ParentSelectionStrategy. A concrete subclass will need to define what percentage of the next population comes from what available population and based on which selection mechanism. The subclass must be registered before it is invoked to create the next generation from the current population and/or the offspring pool.

Finally, class Evolution is the orchestrator of the evolutionary process: the general logic of evolution is implemented here. Any implementation based on the framework will create a concrete subclass of this class and include a main method, so it can be called as a Java application. All above-mentioned registrations of data types, strategies and pool sizes need to be defined in the concrete subclass. Once all necessary elements are registered, the class can be executed to launch the evolutionary process.

2.3 Plug-in Application

The CodeMonkey application is built on top of the Eclipse platform. It uses Eclipses plug-in architecture [10] to create a GUI-based application. The user of CM employs a GUI to provide customizing inputs reflecting a specific EA flavor and target application. Hence, the CM application uses the Eclipse JDT (Java Development Tools) API to create the necessary code, which in turn completes the CM framework. The user can then launch the generated Java program in Eclipse.

Two types of users can use CM to customize and generate an EA in Java: novice and expert. Expert users can directly work with the framework by using existing functionalities or extending them and adding new implementations. Novice users are asked to provide the CM with customizing inputs, which allows CM to generate a Java program that implements a specific EA flavor for a specific EA application. The only part of CM that necessitates the provision of either (1) actual code or (2) a link to an external program or function is the fitness function.

As shown, once the CM plug-in application is launched, the first step is defining the genotype, followed by configuring initialization. Hence, the user defines how fitness will be calculated. This is the only step that necessitates manual code entry or external communications. The next step is defining the termination criteria. This is followed by customizing parent selection. The remaining two steps are defining how variation operations are applied and how the next generation is created. Once those steps are completed, the generated code is compiled and can be run.

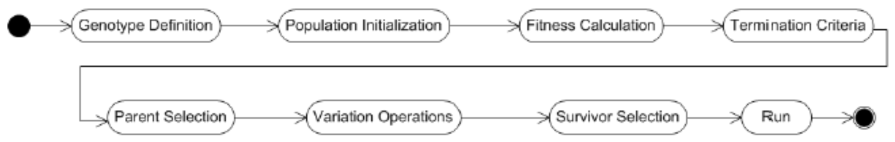

Fig. 2. Activity Diagram of CM Plug-in Application.

2.4 Program Execution

Whether the code is generated by the CM application or directly entered into the generated program, the execution of the resulting program follows the process exhibited in figure 3.

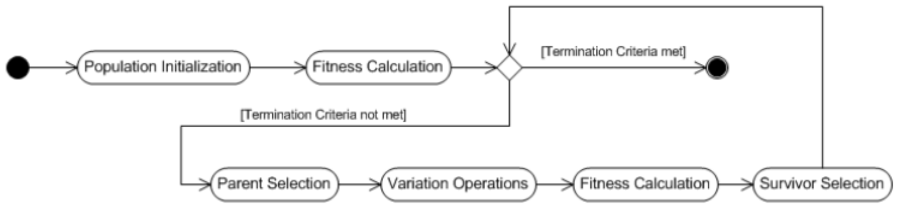

Fig. 3. Activity Diagram of Code Execution.

First comes initialization of the first generation, followed by fitness calculation. Hence, the termination criteria are evaluated. As long as the termination criteria are not satisfied the process goes through parent selection, application of variation operations (to generate offspring), evaluating the fitness of the offspring and hence, generating the next population.

3 Example

In this section, we demonstrate how an end-user can use CM to implement an EA solution to a specific multi-dimensional optimization problem.

The Ackley problem [11] is a n-dimensional minimization problem. The goal is to find $\boldsymbol{x} = [x_1, x_2, ..., x_n]$ within $x_i \in \{-32.768, 32.768\}$ that minimizes the function:

$$F(x) = -20.\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}.\sum_{i=1}^{n}\cos(2\pi.x_i)\right) + 20 + e \quad (1)$$

3.1 Solution

To outline the solution we start by defining the genotype. In this case it will be a list of real-valued numbers, one per dimension. The dimensions can be initialized randomly to values from a limited range. The fitness function is the formula itself. The termination criteria are a combination of goal achieved, evolutionary stagnation and resource exhaustion. For parent selection and survivor selection many types of deterministic and probabilistic selection methods can be selected. To generate offspring, a number of crossover and mutation operators can be used. We configured CM three different ways, with details of the first (reference) configuration presented in Table 2; the difference between the other configurations and the reference is described as well.

3.2 Implementation

The following table presents a comprehensive summary of the parameters and their selected settings for CMs reference configuration C1.

Table 2. List of Parameters and Their Values for the First Configuration (C1) of CM

Genotype Definition	Length	10
	Type	Real
	Lower Bound Value	-32.0 (the same for all genes)
	Upper Bound Value	32.0 (the same for all genes)
	Selected Variation Operators	(Discrete Recombination, Continuous Recombination, Convex Recombination, Local Crossover, One-Position Mutation, Creep Mutation)
Population Initialization	Population Size	300
	Random Generator	Uniform
Fitness Calculation	Mechanism	Internal (formula entered manually)
	Type	Minimization
	Target Fitness	0.0
Termination Criteria	Goal Achieved	Stops if Target Fitness is reached
	Stagnation Reached	Stops if no progress over 1000 generation
	Resource Exhausted	Stops if generation reached 3000
Parent Selection	Window Input Size	20
	Window Output Size	15
	Type of Selection	Proportional (w. replacement)
	Fitness Transformation	Ranking
	Parent Pool Size	150
Variation Operations	Offspring Pool Size	150
Survivor Selection	Selection Type	Proportional (w.o. replacement)
	Elitism (%)	5%
	Re-initialization (%)	5%

C2 alters population size to 100 and offspring pool size to 50; C3 makes parent selection deterministic instead of the original probabilistic mode in C1.

3.3 Results

The best result (owing to configurations C1 and C2) was a fitness value of 8.88×10^{-16}, which is practically zero. The genotype of the best individual is $\{4.9 \times 10^{-324}, 4.9 \times 10^{-324}, 4.9 \times 10^{-324}, 4.9 \times 10^{-324}, 4.9 \times 10^{-324}, 4.9 \times 10^{-324}, 4.9 \times 10^{-324}, 4.9 \times 10^{-324}, 4.9 \times 10^{-324}, 4.9 \times 10^{-324}\}$, which is vector 0. This individual was found at generation 511 (in case of C1) and at generation 4421 (in case of C2). Configuration C3 failed to return an optimal solution even after 7000 generations (which was set as a stop condition). A time course for the evolution of best fitness for the three configurations is presented in figure 4.

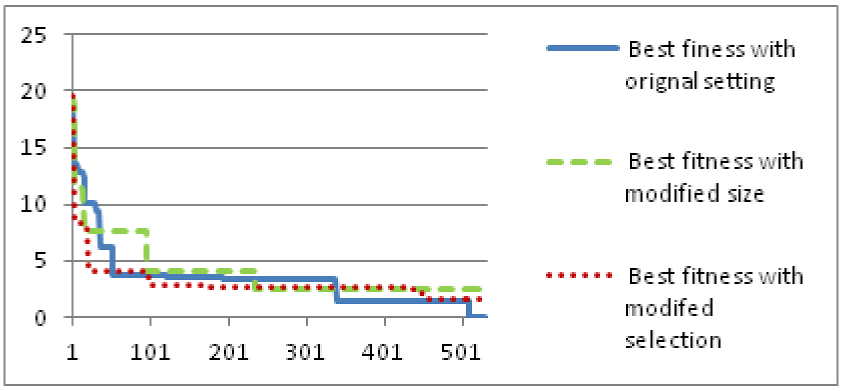

Fig. 4. The History of Evolution of Best Fitness Over 530 Generations.

The flexibility of the framework allows the user to go back to any of the steps and change the settings for that particular part of the evolutionary process. A re-run of the code will incorporate the changes and affect the results. The following table shows the results of the three different configurations: C1, C2 and C3.

Table 3. Results for Different Configurations

	C1	C2	C3
Best Fitness @ Gen. 500	1.3860	2.4721	1.6011
Best Achieved Fitness (& When)	8.88×10^{-16} (@ Gen. 511)	8.88×10^{-16} (@ Gen. 4421)	2.79×10^{-1} (@ Gen.7000)

The first row presents the best fitness achieved by a particular configuration at generation 500. The second row contains the best fitness achieved after 7000 generations or less- that is if the optimal solution is found early.

The second column has the results of reference configuration C1. The results of C2 as shown in the third column of Table 3 demonstrate that a reduction of both the population size and offspring pool by 2/3 lead to an 8 fold increase, compared to C1, of the time necessary to reach the optimal solution. The final configuration (C3) has the same population size and offspring pool size as C1, but it employs deterministic instead of C1s probabilistic parent selection. As a result, C3 achieves a higher best fitness than C2 does initially, but eventually returns a worse best fitness value than both C1 and C2.

In all cases, the manner in which evolution progressed over time was typical and the way in which the results differed was explainable (e.g., a strictly deterministic selection method performing badly on highly multi-modal fitness landscapes).

4 Conclusion

CodeMonkey provides a flexible and easy way to customize and complete (with a fitness function) a generic evolutionary algorithm, to generate a customized Evolutionary Algorithm that reflects the users target application as well as his preferred EA style and configuration. The combination of feature-rich Java and Eclipses popularity make CodeMonkey a handy tool for both expert and non-expert developers of EA applications.

References

1. Genetic and Evolutionary Algorithm Toolbox for Matlab `http://www.geatbx.com`
2. Geatbx Parameter Optimization `http://www.geatbx.com/docu/algindex-09.html#P1058_123869`
3. Evolving Objects (EO) `http://eodev.sourceforge.net`
4. Back, T., Schoenauer, M., Sebag,M., Eiben, A., Merelo, J., Fogarty, T.: A Distributed Resource Evolutionary Algorithm Machine (DREAM). In: IEEE Transaction on Evolutionary Computation, vol. 2, pp. 951–958. (2000)
5. DREAM `http://www.soc.napier.ac.uk/~benp/dream/dream.htm`
6. Watchmaker Framework `http://watchmaker.uncommons.org`
7. Java Genetic Algorithm Package `http://jgap.sourceforge.net`
8. Geatbx Pricing `http://www.geatbx.com/prices.html`
9. Dumitrescu, D., Lazzerini, B., Jain, L., Dumitrescu, A.: Evolutionary Computation, ch. 3–5, 2000
10. Notes on the Eclipse Plug-in Architecture `http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html`
11. Bck, T.: Ackley's Function, in Evolutionary algorithms in theory and practice. Oxford University Press, pp. 142–143. (1996)