An Experimental Evaluation of the
Generic Evolutionary Algorithms Programming Library

Zoltán Tóth * and Gabriella Kókai **

*Department of Informatics, University of Szeged, Hungary (zntoth@inf.u-szeged.hu)
Now visiting:
**Department of Computer Science II, Friedrich-Alexander University of Erlangen-Nürnberg
(kokai@informatik.uni-erlangen.de)

Abstract. In this paper the Generic Evolutionary Algorithms Programming Library (GEA) system is evaluated via a comparison with other genetic programming libraries based on test functions. The purpose of the GEA system is to provide researchers with an easy-to-use and extendable programming library which can solve optimization problems by means of evolutionary algorithms. GEA is implemented in the ANSI C++ programming language and the class hierarchy is designed in a way that enables users to integrate new methods easily. Since there exist several evolutionary algorithm implementations, it is important to check whether it is worth using GEA or not. Besides its flexibility, the presented system outperforms other EA tools on most test functions. ***

Keywords. Evolutionary Computation, Programming Toolkit

Introduction

Engineering applications provide a wide range of optimization problems for people working in this area. The different tasks require in many cases different programming environments to achieve the best results.

The purpose of the Generic Evolutionary Algorithms Programming Library system1 is to provide researchers with an easy-to-use, widely applicable and extendable programming library which solves these tasks by means of evolutionary algorithms [10][12][16].

Evolutionary algorithms are general purpose function optimization methods which search for optima by making potential solutions compete for survival in a population. The better a potential solution is, the better chance it has to survive. The search space is explored by modifying these potential solutions by genetic operators observed in nature: generally mutation and recombination [18].

Evolutionary algorithms have (among others) the following two advantages over other optimization methods: First, in many cases they converge to global optima, and second, the usage of the black-box principle (which only requires knowledge about a function’s input and output to perform optimisation on it) makes them easily applicable to functions whose behaviour is too complex to handle with other methods.

The GEA system contains algorithms for various evolutionary methods, implemented genetic operators for the most common representation forms for individuals, various selection methods, and examples on how to use and expand the library. The implemented genetic operators, selection methods and evolutionary algorithms make the system easy-to-use even for beginners: If the user wants to solve a problem with GEA and the search space consists of, say, bit-strings or real vectors, then he/she only has to implement the problem-specific fitness function, set the parameters of the algorithm and start searching for the solution. GEA is implemented in the ANSI C++ programming language and the class hierarchy is designed in a way that enables users of the system to easily add new selection methods, representation forms for individuals or even evolutionary algorithms.

One must admit that there exist a nice amount of programming libraries that deal with the problem of kinds of evolutionary algorithms [8][9][14][21]. GEA tries to be the ‘alloy’ of these libraries in a manner that it contains several methods and representation forms, so

***This work was supported by the grants of the Bayerischer-Habilitationsförderpreis 1999, DAAD and Siemens AG

1http://gea.ztoth.net
it can be used to solve a large amount of problems. Although there are functions in GEA (such as algorithms for evolutionary strategies (ESs), the so-called meta-ES and the adaptation of the probability of the genetic operators [11]) which are supported only by a few other libraries. In this paper an evaluation of GEA is given making comparison with other genetic programming libraries based on carefully selected test functions. The executed runs show that GEA performed very well on the test suit with regard to execution speed and achieved fitness values as well. The presented results show empirical evidence that the developed system can expect success in the field of applications.

In the following, Section 1 offers an overview of evolutionary algorithms. Section 2 contains some details of the GEA system: The class hierarchy and the purpose of the classes. In Section 3 a comparison of GEA and some other systems can be found. The GEA system and the other libraries have been tested on some standard test functions. The results of these tests are presented in this section. Finally in Section 4 a summary of present and future work is given.

1 Evolutionary Algorithms

In this section an overview of evolutionary algorithms is given, focusing on details that are important for the GEA system; that is, the theoretical foundations of the implemented methods are described here.

Evolutionary algorithms (EAs for short) are general purpose function optimization methods which use the 'survival-of-the-fittest'-model known from nature [4]. In this model individuals compete for resources in an environment and selection assures that individuals which are better suited for the given environment will produce more offspring. Thus the preservation of good attributes is guaranteed.

Unlike most optimization methods, EAs consider several potential solutions at a time. These potential solutions, called individuals from now, form a population. The individuals interact with each other, thus they create new individuals to form a new generation.

An individual of the population is represented with a sort of data structure. The most common representation forms for individuals are bit-string and real vector. Each element of the vector is called a gene. The chain of genes is also called a chromosome. The values in it are the individual’s genotype. The appearance of an individual – which can be e.g. a permutation of certain numbers – is called phenotype. Evolutionary algorithms work on the level of the genotype, which means that they modify the encoded form of individuals. When evaluating an individual in its current environment, its phenotype is considered. The result of the evaluation is usually a real number, and the task of the evolutionary algorithm is either to maximize or minimize this number. From the evaluation, a positive fitness value is computed, which is always greater for fitter individuals. This fitness value is considered when performing selection.

The creation of new individuals is done by applying certain genetic operators on the selected parents. The most common genetic operators are reproduction, mutation and recombination. Reproduction simply copies the individual into the new generation, while mutation modify its argument by randomly changing each gene of it with a certain probability. Recombination takes two or more individuals and creates new ones by replacing parts of their gene-chains. Each genetic operator is applied with a certain probability. However, sometimes one operator is more efficient than the others and it is not easy (or at least it requires experiment) to set the probabilities correctly at the start of an evolution process. Davis’s solution is to change the probabilities dynamically during the evolution process by observing the effectiveness of the operators. He calls this method the adaptation of operator probability [5].

Executing an evolutionary algorithm is an iterative process: At the beginning, an initial population is created and its individuals are evaluated. The iteration steps contain the creation of the new population and the evaluation of the newly created individuals. The process stops when a certain halting condition is satisfied, which can be, for example reaching a given generation number.

Several kinds of evolutionary algorithms are known, from which the most important ones are genetic algorithms (GAs) [6][10] and evolutionary strategies (ESs) [16]. They were developed independently in the 1970s: GAs were introduced by John Holland and analyzed by his students (e.g. Kenneth De Jong) in the USA, and at the same time, evolutionary strategies were invented in Germany by Ingo Rechenberg. The main differences between these two kinds of EAs are the method of creating the new generation and the typical representation form for individuals. The typical representation form for individuals is bit-string for GAs and real vector for ESs.

The two kinds of EAs also differ in the manner in which genetic operators are applied. Genetic algorithms use a wide range of selection methods to select the individuals that can reproduce. These selection methods apply different selection pressure. The recombination operator of GAs always takes two individuals and produces two descendants, that’s why it is often called crossover. The most widespread recombination method is to select one or more recombination points on the chromosome and exchange the parts of
the individuals between these points. Another possibility is the so-called parametric uniform crossover, where each gene is exchanged with a certain probability.

There is a special kind of genetic algorithms, namely genetic programming (GP) introduced by John R. Koza[12]. The main invention of GPs is that branching structures can be evolved. Most methods are the same as in GAs, but there are special genetic operators designed for these structures, e.g. recombination replaces subtrees of the selected individuals.

Evolutionary strategies always use best and random selection. The recombination operator always produces one descendant from the parent individuals and can be discrete or intermediate. At discrete recombination the gene values of the new individual are set from a randomly selected parent, and at intermediate recombination the offspring’s gene values are computed by averaging the parent individuals’ appropriate gene values.

Evolutionary strategies can be classified into the so-called plus and comma strategies. In short, the difference between the two strategies is that when the comma strategy is used, parents die off after creating their offspring. In the case of the plus strategy, parents compete with their offspring for survival. The model of competing subpopulations can be applied by the so-called meta-ES method [11].

2 The GEA System

This section describes the GEA system in detail. It is explained how the idea of creating such a programming library came up. Then the class hierarchy of the latest version is presented. This means that the implemented representation forms, their genetic operators and the evolutionary algorithms are also mentioned here.

The first aim of the GEA system was to provide a basis for the evolution of Lindenmayer systems (L-systems for short) [13]. These structures are capable of describing fractal structures such as trees (see the Tevol program, [19]) or even the blood vessels of the human retina (the GREDEA project, [20]). These two applications required the evolution of the rewriting rules of the L-systems as well as their parameters. The most suitable evolutionary algorithms to evolve the rewriting rules and the parameter vectors are genetic programming and evolutionary strategies, respectively. A suitable C++ programming library which dealt with GPs and ESs at the same time could not be found – and the design and implementation of GEA has began.

The class hierarchy of the current version of GEA can be seen in Figure 1. The names of the abstract classes are written in italics. Class Evolvable is the superclass of all evolvable objects. The classes SelectionMethod and NextGenMethod define interfaces for selection methods and evolutionary algorithms. The abstract classes enable the user of the system to easily integrate new functions into GEA by writing so-called plug-in modules which are loaded at running time into the system. The advantage of the plug-ins is that the software can be extended without changing its implementation or recompiling it. The plug-in classes are denoted by a ‘+’ sign in the class hierarchy.

```plaintext
Evolvable
- EvolvableBitString +
- EvolvableRealVector +
- Population
EA
NextGenMethod
- GANextGen +
- ESNextGen +
SelectionMethod
- ProportionalSelection +
- BoltzmannSelection +
- RankSelection +
- TournamentSelection +
- BestSelection +
- RandomSelection +
- SigmaSelection +
```

This data structure is designed to contain parameters of arbitrary processes or systems. It is easy to define the types and restrictions of parameters, and even relations between them. If an extension of GEA requires new parameters to be added, then they must be entered into the system’s parameter structure definition file whose syntax is given by a set EBNF rules. The parameter values can be set using a graphical user interface or directly in the input files. The GEA system is designed in a way that the evolutionary parameters can be modified even during a started evolutionary process.

Class Evolvable is the abstract superclass of all evolvable classes: It declares all the functions a class has to have in order to become an evolvable class. The standard GEA system contains two individual representations: EvolvableBitString and EvolvableRealVector are classes which implement the genetic operators of these two individual representations. If the possible solutions of a problem can be represented by bitstrings or real valued vectors, then only the fitness function has to be implemented and passed to the constructor of the selected class as a callback function.

The genetic operators are implemented according to the representation form. For bitstrings, mutation can change a bit by either flipping it or generating a random bit into its place. The GA-crossover can be single-point, multi-point or parametrised uniform crossover. Recombination of the ES works by getting genes from the selected parents (only discrete recombination is applicable, since it does not make sense to compute the average of bits).
For real vectors, the mutation of a gene can be done by adding a Gaussian random number to it or multiplying it with a randomly generated value. Crossover is the same as at bitstring representation, and recombination can be either local/global and discrete/intermediate.

The system can optionally adapt the probability of the genetic operators, that is, it can observe their effectiveness and change the probabilities according to the result. This feature can be useful to set up the appropriate operator probabilities.

Class Population represents a population (a community of individuals) in the GEA system. The pointers to the individuals are stored in an array and are sorted by decreasing fitness values. The class has some functions which perform preprocessing computations needed by certain special selection methods.

Being a subclass of Evolvable, populations of populations can be created, thus populations can be evolved, too. This makes meta-ES available in the system. The implementation of the genetic operators is very similar to that of bitstrings. The fitness value of a population can be either its best individual’s fitness value or the mean of all individuals’ fitness values.

The selection operators in GEA are all implemented as subclasses of the abstract class SelectionMethod. Evolutionary strategies always use random selection, but genetic algorithms and other evolutionary algorithms which might be added to the system by its users can use arbitrary selection methods. The required method can be specified among the parameters and the system loads the appropriate plug-in when the evolutionary process is created.

Class EA represents an evolution process in the GEA system. It has all methods that are necessary to handle a population and create new generations from it. It has to know the parameters of the evolutionary algorithms and the representation type of the individuals. After creating an EA object, only its NextGen function has to be called to run the evolution process. Eventual errors caused by incorrect parameter setting or insufficient system resources are handled with a general error handling procedure.

3 Comparison with Other Programming Libraries

In this section a comparison of the GEA system with some other freely available evolutionary/genetic programming libraries is given. There are a large number of programming libraries which try to deal with evolutionary algorithms, but most of them are written in C or other programming languages, thus the advantages of C++’s object-oriented capabilities cannot be exploited. For such libraries, see SGA-C [8] and GENESIS [9]. Another problem with existing programming libraries is that some of them are capable to work only with bitstring and real vector individual representations (GENESIS). Since the GEA system is written in ANSI C++ language and – thanks to the plug-in technology – can be applied on any representation forms of the individuals, these deficiencies are overcome.

There are other programming libraries (e.g. GAlib [21]) written in C++ with support for any representation forms, but even these libraries do not contain methods for evolutionary strategies. The most important thing in the GEA system is that at the time when its development started, there could not be found any programming libraries with implementation of ESs. Thus, experiments with them could not be done. Besides the implementation of functions needed for ESs, the GEA system also provides functions for experiments with meta-ES as well.

A larger project, the EO Evolutionary Computation Framework [14] exists supervised by J. J. Merelo at the University of Granada that deals with any kind of evolutionary algorithms, but ESs were not implemented in the ancestor of this system, GAGS.

The above mentioned evolutionary systems and GEA were tested on several test functions. These test functions were chosen according to [7], so that they differ in their modality, separability and regularity (that is, the regular or irregular arrangement of the local optima). Table 1 shows the exact definition and the attributes of the functions. The dimensionality is not indicated in the table: \( n \) is 30 in all cases, i.e. \( \mathbf{x} = (x_1, x_2, \ldots, x_{30}) \).

Since multi-modal and inseparable functions mean more severe challenge to evolutionary algorithms, these are represented with greater weight in the test suite. Moreover, it is also important to make a difference between regular and irregular functions. The local optima of the Ackley and Griewangk functions are distributed normally, while the Fletcher-Powell and Langerman functions are based on random values, thus these are irregular.

All test functions were implemented in all systems, and ten independent runs were performed on each function. The running times were measured, thus the convergence speed can also be compared. When it was possible, the parameters of the evolution processes were set to the same values in all cases. These common parameters were the following:

- Population size: 100
- Selection method: Fitness proportional
- Mutation probability: 0.01
- Recombination probability: 0.8
- For each test function, the number of processed generations was determined according to [7]
When it was possible, the elitism rate was set to 1 and the mutation rate to 2.8. Note that the Fletcher-Powell function had to be minimized and some systems (EO, SGA-C) are not able to perform minimization on the target function. The fitness computation had to be modified in the case of these systems and thus a fair comparison could not be made.

Table 2 shows the running times of the different genetic systems for each of the test functions. The entry of the fastest system is typed in boldface and the values in parentheses show the speed factor of the respective system to the fastest one. It can be observed that the ‘best’ library with respect to execution speed is GEA. It was the fastest on four of the seven test functions and reached a second place on the other three functions. The second fastest system was GENESIS which outperformed GEA on the more complicated test functions.

Of course when observing the performance of an evolutionary system, one must not regard only the execution speed. The acquired fitness values are more important than the speed of the process. The best fitness values found by the various systems are shown in Table 3 with the best values typed in boldface. To make the comparison on the single test functions easier, certain number of points were assigned to each library for each function. These values are indicated in parentheses in the table. Where the goal was to maximize the target function, ten points were assigned to the library which achieved the highest target value. The number of points given to the other libraries is proportional to the fitness value achieved by the respective system. For the Fletcher-Powell function, where the goal was to minimize the target function value, the values in the parentheses show the factor between the result achieved by the according library and the result of the best system.

On four test functions \( f_1, f_2, f_5, f_7 \), GEA achieved the best fitness values and in the case of three of these functions \( f_1, f_2, f_5 \), it was the fastest system as well. In the case of \( f_7 \), GENESIS was the fastest implemen-

\[ f_1(x) = \sum_{i=1}^{n} x_i^2, \text{where} -5.12 \leq x_i \leq 5.12 \ (i = 1, 2, \ldots, n) \]

\[ f_2(x) = \sum_{i=1}^{n} \left( \frac{x_i}{2^n} \right), \text{where} -5.12 \leq x_i \leq 5.12 \ (i = 1, 2, \ldots, n) \]

\[ f_3(x) = 10n + \sum_{i=1}^{n} \left( x_i^2 - 10 \cos(2\pi x_i) \right), \text{where} -5.12 \leq x_i \leq 5.12 \ (i = 1, 2, \ldots, n) \]

\[ f_4(x) = 20 + e - 20 \exp \left( -0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \right) - \exp \left( \frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i) \right), \text{where} e = \exp(1) \text{ and } x_i \leq 20 \ (i = 1, 2, \ldots, n) \]

\[ f_5(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{1000} + \prod_{i=1}^{n} \cos \left( \frac{\sqrt{2} x_i}{\sqrt{n}} \right), \text{where} -600 \leq x_i \leq 600 \ (i = 1, 2, \ldots, n) \]

\[ f_6(x) = \sum_{i=1}^{n} (A_i - B_i)^2, \text{where} A_i = \sum_{j=1}^{n} (a_{ij} \sin x_j + b_{ij} \cos \alpha_j), \]
\[ B_i = \sum_{j=1}^{n} (a_{ij} \sin x_j + b_{ij} \cos \alpha_j), \text{and} -\pi \leq x_i \leq \pi \ (i = 1, 2, \ldots, n) \]

\[ f_7(x) = -\sum_{i=1}^{n} c_i \exp \left( -\frac{1}{\pi} \sum_{j=1}^{n} (x_j - a_{ij})^2 \right) \cdot \cos \left( \pi \sum_{j=1}^{n} (x_j - a_{ij})^2 \right), \text{where} m = 30 \text{ and } 0 \leq x_i \leq 10 \ (i = 1, 2, \ldots, n) \]

<table>
<thead>
<tr>
<th>Notation</th>
<th>Name</th>
<th>Definition</th>
<th>Modality</th>
<th>Separable?</th>
<th>Regular?</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>Sphere model [17]</td>
<td>( f_1(x) = \sum_{i=1}^{n} x_i^2, \text{where} -5.12 \leq x_i \leq 5.12 \ (i = 1, 2, \ldots, n) )</td>
<td>uni</td>
<td>yes</td>
<td>N/A</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>Schwefel's double sum [17]</td>
<td>( f_2(x) = \sum_{i=1}^{n} \left( \frac{x_i}{2^n} \right), \text{where} -5.12 \leq x_i \leq 5.12 \ (i = 1, 2, \ldots, n) )</td>
<td>uni</td>
<td>no</td>
<td>N/A</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>Generalized Rastrigin’s function [2]</td>
<td>( f_3(x) = 10n + \sum_{i=1}^{n} \left( x_i^2 - 10 \cos(2\pi x_i) \right), \text{where} -5.12 \leq x_i \leq 5.12 \ (i = 1, 2, \ldots, n) )</td>
<td>multi</td>
<td>yes</td>
<td>N/A</td>
</tr>
<tr>
<td>( f_4 )</td>
<td>Generalized Ackley’s function [1]</td>
<td>( f_4(x) = 20 + e - 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), \text{where} e = \exp(1) \text{ and } x_i \leq 20 \ (i = 1, 2, \ldots, n) )</td>
<td>multi</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>( f_5 )</td>
<td>Generalized Griewank function [2]</td>
<td>( f_5(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{1000} + \prod_{i=1}^{n} \cos \left( \frac{\sqrt{2} x_i}{\sqrt{n}} \right), \text{where} -600 \leq x_i \leq 600 \ (i = 1, 2, \ldots, n) )</td>
<td>multi</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
| \( f_6 \) | Fletcher-Powell function \[1\] | \( f_6(x) = \sum_{i=1}^{n} (A_i - B_i)^2, \text{where} A_i = \sum_{j=1}^{n} (a_{ij} \sin x_j + b_{ij} \cos \alpha_j), \]
\[ B_i = \sum_{j=1}^{n} (a_{ij} \sin x_j + b_{ij} \cos \alpha_j), \text{and} -\pi \leq x_i \leq \pi \ (i = 1, 2, \ldots, n) \) | multi | no | no |
| \( f_7 \) | Generalized Langerman function \[3\] | \( f_7(x) = -\sum_{i=1}^{n} c_i \exp \left( -\frac{1}{\pi} \sum_{j=1}^{n} (x_j - a_{ij})^2 \right) \cdot \cos \left( \pi \sum_{j=1}^{n} (x_j - a_{ij})^2 \right), \text{where} m = 30 \text{ and } 0 \leq x_i \leq 10 \ (i = 1, 2, \ldots, n) \) | multi | no | no |

Table 1 The definition of the test functions

\[ a_{ij}, b_{ij} \in \{-100, \ldots, 100\} \ (i, j = 1, 2, \ldots, n) \text{ are random integers, and} \ c_{ij} \in [-\pi, \pi] \ (j = 1, 2, \ldots, n) \text{ is the randomly chosen global optimum position. Reference values for matrices} \ A, B \text{ and vector} \ \alpha \text{ can be found in [1].} \]

\[ \text{The matrix} \ A = a_{ij} \ (i, j = 1, 2, \ldots, n) \text{ and vector} \ \alpha \text{ are randomly generated over the set of real numbers. [7] contains reference to these values; they also can be found at http://www.wi.leidenuniv.nl/CS/ALP/alea.html.} \]
Table 2  The running times of the libraries on each of the test functions

<table>
<thead>
<tr>
<th>Genetic Library</th>
<th>$f_1$ (s)</th>
<th>$f_2$ (s)</th>
<th>$f_3$ (s)</th>
<th>$f_4$ (s)</th>
<th>$f_5$ (s)</th>
<th>$f_6$ (s)</th>
<th>$f_7$ (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGA-C</td>
<td>8.35 (1.8)</td>
<td>141.96 (6.4)</td>
<td>91.66 (1.7)</td>
<td>17.50 (1.6)</td>
<td>16.04 (1.8)</td>
<td>1084.87 (1.5)</td>
<td>463.87 (2.9)</td>
</tr>
<tr>
<td>GENESIS</td>
<td>5.78 (1.3)</td>
<td>56.85 (2.1)</td>
<td>56.10 (1.0)</td>
<td>11.21 (1.0)</td>
<td>12.51 (1.4)</td>
<td>376.25 (1.0)</td>
<td>159.74 (1.0)</td>
</tr>
<tr>
<td>GALib</td>
<td>6.46 (1.4)</td>
<td>210.65 (7.8)</td>
<td>88.90 (1.6)</td>
<td>22.88 (2.0)</td>
<td>13.83 (1.5)</td>
<td>3045.32 (4.1)</td>
<td>1221.76 (7.6)</td>
</tr>
<tr>
<td>EO</td>
<td>7.54 (1.6)</td>
<td>59.49 (2.2)</td>
<td>85.09 (1.6)</td>
<td>36.86 (3.5)</td>
<td>14.10 (1.5)</td>
<td>736.25 (1.0)</td>
<td>544.61 (3.4)</td>
</tr>
<tr>
<td>GEA</td>
<td>4.61 (1.0)</td>
<td>26.85 (1.0)</td>
<td>53.82 (1.0)</td>
<td>13.69 (1.2)</td>
<td>9.16 (1.0)</td>
<td>1077.09 (1.5)</td>
<td>417.50 (2.6)</td>
</tr>
</tbody>
</table>

Table 3  The best fitness values found by the systems for each test functions

<table>
<thead>
<tr>
<th>Genetic Library</th>
<th>$f_6$ (s)</th>
<th>$f_7$ (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGA-C</td>
<td>834800 (88)</td>
<td>0.1072 (2.9)</td>
</tr>
<tr>
<td>GENESIS</td>
<td>9453 (40)</td>
<td>0.1846 (4.9)</td>
</tr>
<tr>
<td>GALib</td>
<td>385497 (41)</td>
<td>0.0145 (0.4)</td>
</tr>
<tr>
<td>EO</td>
<td>29934 (3)</td>
<td>0.3416 (9.2)</td>
</tr>
<tr>
<td>GEA</td>
<td>44937 (5)</td>
<td>0.3731 (10)</td>
</tr>
</tbody>
</table>

As a summary, GEA proved to be the ‘best’ system with respect to achieved fitness values: It found the best individuals in four of the seven test cases, finished on the second and the third place twice and once, respectively. GENESIS performed nearly as well as GEA and the other three libraries had results of different quality on the test functions.

**Figure 2**  The achieved fitness values and running times for the Sphere model ($f_1$)

**Figure 3**  The achieved fitness values and running times for Schwefel’s double sum function ($f_2$)

**Figure 4**  The achieved fitness values and running times for the generalized Rastrigin’s function ($f_3$)

Figures 2 through 8 show the performance of the different programming libraries for each test function. The achieved best fitness value is plotted against the
elapsed time, so it is quite easy to compare the systems (e.g. their real problem solving speed). The non-monotony of the graphs of GENESIS and EO shows that these libraries do not use elitism to preserve the best individual found so far.

\[ f_2, \ldots, f_7 \] contain similar information about the speed of the systems and the best individuals found, so these observations are not detailed. These are the data summarized in Tables 2 and 3. Additional system-specific observations can be made from these graphs: For example, GENESIS and GALib start with relatively bad individuals and the progress is continuous during the evolution process, while GEA finds pretty good solutions even in the first generation, and the progress is staggered, with long stagnant periods.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5}
\caption{The achieved fitness values and running times for the generalized Ackley's function ($f_4$).
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6}
\caption{The achieved fitness values and running times for the generalized Griewangk function ($f_5$).
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure7}
\caption{The achieved fitness values and running times for the Fletcher-Powell function ($f_6$).
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure8}
\caption{The achieved fitness values and running times for the generalized Langerman function ($f_7$).
\end{figure}

\section{Summary and Future Work}

In this document the GEA (Generic Evolutionary Algorithms) system, an evolutionary algorithms programming library written in the ANSI C++ programming language is compared to other available and wide-spread evolutionary libraries.

The design and implementation of GEA was started with projects that evolve parametric Lindenmayer systems to describe branching structures. These evolution processes required the use of genetic programming and evolutionary strategies, and at that time there was no programming libraries that provided both of these two algorithms.

The GEA system in its present phase contains implementations of genetic algorithms and evolutionary strategies, several types of selection methods, and genetic operators for bitstring and real-vector individual representations. The examples provided with the system and the current applications contain the genetic operators of permutations and parametric L-systems as well. Genetic programming can be applied by the creation of tree-like individual representation forms. The library is designed to be modular, that is, it can be easily extended by subclassing the respective abstract classes to create plug-ins.

The comparison of the GEA system with other libraries with a similar goal has proven that the en-
ergy invested into the development of the programming library was not wasted. It is not only a flexible and easy-to-use library with simple ways of application and extension, but its performance is very good on many problems of different characteristics. Despite of the low level of optimization in the genetic functions (the system is continually under development and being optimized), it is faster than other freely available evolutionary systems. The numerous parameters make the fine-tuning of the evolution process available in order to achieve the best possible solutions in affordable time.

The current implementation of GEA will be extended with the following features in the foreseeable future:

- More optimization to increase the speed of the system.
- Coevolution [15] will be made available by letting the individuals know about their mates in the population. This is not implemented in any of the libraries mentioned in Section 3.
- A graphical user interface is under development for GEA which will enable the user to set the genetic parameters easily, to observe the performance of an EA run, and to display the individuals graphically. The GTK GUI Toolkit is used in the implementation. GEA stays independent of the GUI, that is, it will be possible to use the system without it.
- A utility will be designed and implemented which will help the user to design evolutionary algorithms by the drag-and-drop technique.

It can be seen that the GEA system in its present state is a widely applicable and easy-to-use library. As many research projects, it is of course constantly under development. Among the above mentioned improvements other modifications also will be carried out as further reports and comments arrive from the researchers working in the field.

References


http://www.gtk.org