eaLib

User's Manual

Version 0.2.4

Author:

Andreas Rummler

Date:

January 7, 2002

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimization problems can be found in a large number of areas. Most of these problems have a huge search space, so finding an optimal solution in a analytical way is impossible in most of these cases. The optimization task must be tackled in a different way. Several approaches and algorithms exist, both deterministic and non-deterministic. A popular stochastic approach are algorithms belonging to the family of Simulated Annealing. Another from the group of non-deterministic ones are algorithms based on simulated evolution. This kind of algorithms are discussed in this introduction.

1.1 Definitions

To provide correct informations in this manual, some definitions are necessary. Throughout *eaLib* five data types are used. The range of values are defined according to the Java Language Specification ([GJS96]). The definitions are given in table 1.1 below. If one of these datatypes is used in this manual the appropriate range of values is valid.

Type	Minimum Negative	Maximum Negative	Size $[bit]$
boolean	n/a	-1	1
int	-2147483648	-1	32
long	-9223372036854775808	-1	64
float	$-3.40282347 \cdot 10^{38}$	$-1.40239846 \cdot 10^{-45}$	32
double	$-1.7976931348623157 \cdot 10^{308}$	$-4.94065645841246544 \cdot 10^{-324}$	64
	Minimum Positive	Maximum Positive	
boolean	n/a	0	1
int	0	2147483647	32
long	0	9223372036854775807	64
float	$1.40239846 \cdot 10^{-45}$	$3.40282347 \cdot 10^{38}$	32
double	$4.94065645841246544 \cdot 10^{-324}$	$1.7976931348623157 \cdot 10^{308}$	64

Table 1.1: Valid Java data types.

Chapter 2

Installation

This chapter briefly describes the installation procedures of the binary and the source release.

2.1 Requirements

Before installing the *eaLib*-Package, make sure that the following requirements are met:

- Java Development Toolkit 1.3 [JDK], JDK 1.2.2 should also work, but has never been tested
- Java Serialization to XML [JSX]
- Log4J Logging Toolkit [Log]
- Xerxes Java Parser [Xer], currently unused, but future XML handling will rely on this parser
- (optional) Ant [Ant], for building the source release
- (optional) JUnit [JUn], for building the source release and running test cases

2.2 Installing the Binary Release

To install *eaLib* first depack the archive to a directory of your choice:

```
mkdir java
mv ealib-<version-number>.tar.gz java
gunzip ealib-<version-number>.tar.gz
tar xvf ealib-<version-number>.tar
```

This creates a subdirectory called 'ealib'. All necessary jar-Archives can be found in the 'dist' subdirectory. Now simply add all jar-Files to your CLASSPATH environment variable. Now it is possible to compile own programs or run the examples.

2.3 Installing the Source Release

To compile and install the source release depack the archive to a directory of your choice:

```
mkdir java
mv ealib-src-<version-number>.tar.gz java
gunzip ealib-src-<version-number>.tar.gz
tar xvf ealib-src-<version-number>.tar
```

This creates a subdirectory called 'ealib'. For building *eaLib* it is required to install all the mentioned software packages from above. Please refer to the documentation of the tools for installation instructions. As a build sytem the tool *Ant* is used. *Ant* relies on a build file ('build.xml') which is included in the *eaLib*-distribution. The build file defines several build targets. To compile *eaLib* invoke ant on the target 'compile':

```
ant compile
```

For building the examples and the test cases separately, the targets 'examples' and 'test' exist. The compiled classes are put in the a subdirectory called 'classes'. The API documentation can be generated with invoking ant on the target 'docs'. An overview over existing targets can be received by calling ant with the parameter '-projecthelp':

```
ant -projecthelp
```

Chapter 3

Evolutionary Algorithm Foundations

To be written . . .

Chapter 4

Represention of Possible Problem Solutions

The first step when implementing an evolutionary algorithm is thinking about an appropriate representation of a solution to the problem. In this chapter we will discuss how this step can be done and how a possible implementation can look like.

In biology the genetic information of a creature resp. an individual is contained in its chromosome set. Therefore the chromsomes set is a kind of construction plan for a creature of that species. So an individual used in an evolutionary algorithm also contains a set of chromosomes[1]. If an individual is a possible solution to our optimization problem, this set must be the representation of this problem.

In the first genetic algorithms that were developed by Goldberg [Gol89] and Rechenberg/Schwefel [Rec73] binary string or real-valued numbers were used for representing potential solutions of a given optimization problem. As an example let's take a look at the a threedimensional function given in figure 4.1. Let's imagine this function is to be minimized. With other words we have to find two values for x and y so that $f(x, y)$ gets as small as possible. A possible solution to this problem would be for instance $x = 2.0$ and $y = 3.5$. A genetic representation for this potential problem solution is also given in figure 4.1. The value of x is assigned to a first chromosome and the value of y is assigned to the second chromosome. So the genetic representation contains two chromosomes each holding a float number (or a double if more precision is required).

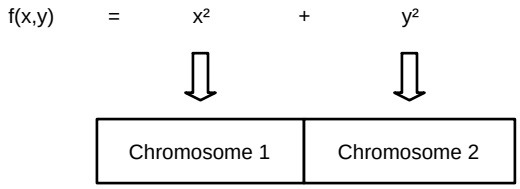

Figure 4.1: Representation of a Problem Solution.

Unfortunatly the genetic representations used in traditional algorithms are often neither very descriptive for the human eye nor very flexible. There is no reason not to use other data types for the task of expressing a problem solution. Therefore *eaLib* uses a more generic model. Each individual can contain an arbitrary number of chromosomes of various data types (figure 4.2).

4.1 Creating Individuals

In this section we learn how to create individuals. An overview over the class structure gives the UML diagram in figure **??**.It can be seen that an instance of the class *Individual* contains exactly one instance of the class

[1] Note that some people refer to the term *gene* in this context instead of chromosomes. The meaning is the same, although in my opinion it is not exact, so I will refer to the term *chromosome*

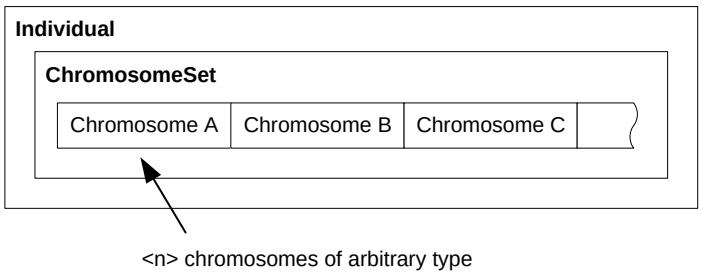

Figure 4.2: Structure of the genetic representation used in eaLib.

ChromsomeSet. The chromosome set itself can contain an arbitrary number of chromosomes, which all have specific type. The types that are already provided by *eaLib* are enumerated in the next section.

The creation of an individual for the example from the last section can be seen in listing XXX.

```
// create a chromosome set
ChromosomeSet mySet = new ChromosomeSet();
// add the chromosomes with initial float values
// add the first chromosome to slot 0 in chromosome set
mySet.add( new FloatChromosome( (float ) 2.0 );
// and add the second chromosome to slot 0 in chromosome set
mySet.add( new FloatChromosome( (float ) 3.5 );
// finally create the individual
Individual myIndividual = new Individual ( mySet );
```

Listing 4.1: creation of an individual

It is quite an easy task to create individuals for the given problem. First an instance of the class *ChromosomeSet* called *mySet* is created. Then two chromosomes holding float numbers are put into the chromosome set. Finally the individual *myIndividual* is created.

Every chromosome set defines the toString() method, which calls all *toString()* methods of all contained chromosomes. The textual representation of the chromosomes is concatenated without whitespaces and separated by commas. The whole string is enclosed by parantheses. List objects are enclosed by brackets and list elements are separated by commas. Calling the *toString()* on our individual would produce the following output:

```
{2.0,3.5}
```

4.2 Predefined Chromosome Types

This section gives a short overview over all predefined chromosome types that come with *eaLib*.

- ArrayChromosome
  Chromosome for arrays of all types. This can be arrays of primitive data types as well as arrays of objects.
- ArrayListChromosome
  Chromosome holding a single array list.
- BinaryStringChromosome
  Chromsome of type binary string. There is a special class *BitVector* to provide the possibility to implement individuals using the traditional binary string representation.
- DoubleChromosome
  Chromosome of type double.

- FloatChromosome
  Chromosome of type float.
- IntegerChromosome
  Chromosome of type integer.
- LinkedListChromosome
  Chromosome containing a linked list.
- ListChromosome
  This is the abstract base class for ArrayListChromosome, LinkedListChromosome and VectorChromosome. It can't be instantiated.
- LongChromosome
  Chromosome of type long.
- NullChromosome
  This is a dummy chromosome containing a null-value. It can be used for test and debug purposes.
- StringChromosome
  Chromosome of type string.
- TreeChromosome
  Chromosome containing a tree.
- VectorChromosome
  Chromosome containing a vector. In most cases the ArrayListChromosome should be prefered over this for performance reasons.

4.3 Creation of new Chromosome Types

For most application cases the predefined chromosome types will work fine. But a user could require a special representation form that is tailored to his task. For that reason there is the possibility to create new chromosome types. This section shows how this can be done.

As an example we will create a chromosome type that contains an integer number and a string (for whatever purpose this this may be good ...). All chromosomes extend the abstract class *Chromosome* in package *mss.ea.chr*. This class has two abstract methods which must be implemented: *clone()* and *equals(Object)*.

Pieces of the listing for our new chromosome class can be seen and are explained step by step below. First we have to create a new class *IntStringChromosome* for our chromosome which extends the abstract base class. The base class has already an instance variable for the chromosome object (*co*), but we will ignore that. Instead we define our own instance variables:

```
public class IntStringChromosome extends Chromosome {
    protected int integerPart ;
    protected String stringPart ;
}
```

Listing 4.2: Chromosome Class with Instance Variables

Next we create a constructor for our class. Note that we have to call the constructor of the base class, but we do not use the predefined chromosome object:

```
public IntStringChromosome( int i , String s ) {
  super( null );
   integerPart = i;
   stringPart = s;
```

```
}
```

Listing 4.3: Constructor

Other classes may want to access both instance variables, so we create two access methods:

```
public int getIntegerPart () {
  return integerPart ;
}

public String getStringPart () {
  return stringPart ;
}
```

Listing 4.4: Access Methods

Now for the important things. We have to implement the two abstact methods of the base class. First is the *equals(Object)* method. This method is used to compare this chromosome to another one. First we try to cast the given object to the type of our chromosome – if that fails both objects can't be equal and we return false. But if this cast succeeds we can test the instance variable and and return true if they are equal.

```
public boolean equals ( Object o ) {
  try {
    IntStringChromosome other = ( IntStringChromosome) o;
    if ( integerPart == other . getIntegerPart () &&
         stringPart . equals ( other . getStringPart () ) ) {
      return true;
    } else {
      return false ;
    }
  } catch ( ClassCastException cce ) {
    return false ;
  }
}
```

Listing 4.5: Equals-Method

Finaaly we have to implement the *clone()* method. This method is used by some genetic operators to create copies of the chromosome to work with.

```
public Object clone () {
  return new IntStringChromosome( integerPart , stringPart );
}
```

Listing 4.6: Clone-Method

This finishes the process of creation of an own chromosome. The full listings of our new chromosome class is shown below.

```
public class IntStringChromosome extends Chromosome {

  protected int integerPart ;
  protected String stringPart ;

  public IntStringChromosome( int i , String s ) {
    super( null );
    integerPart = i;
    stringPart = s;
  }
```

```
12    public int getIntegerPart () {
        return integerPart ;
14    }

16    public String getStringPart () {
        return stringPart ;
18    }

20    public boolean equals ( Object o ) {
        try {
22        IntStringChromosome other = ( IntStringChromosome) o;
          if ( integerPart == other . getIntegerPart () &&
24             stringPart . equals ( other . getStringPart () ) ) {
            return true;
26        } else {
            return false ;
28        }
        } catch ( ClassCastException cce ) {
30        return false ;
        }
32    }

34    public Object clone () {
        return new IntStringChromosome( integerPart , stringPart );
36    }

38  }
```

Listing 4.7: Creation of a Chromosome Class holding an Integer and a String

Chapter 5

Score Evaluation

After we have seen how individuals can be created, now these individuals must be evaluated for thier quality. This chapter introduces the term *score* as a quality appaisal for individuals. Furthermore it is shown how scores are created and assigned to individuals. The last section deals with the comparison of individuals by scores.

5.1 Introducing the Term Score

In most publications the term *fitness* is used as a quality indicator for individuals. This is not wrong, it is just not general enough. In *eaLib* there is a strong differentiation between the terms *fitness* and *score* . The score is another word for the target objective value while the fitness is a relative quality indicator. This chapter deals with the score, the fitness and its evaluation is introduced in the next chapter.

By looking at the example from the last chapter, it is quite clear what the target objective value is. It is just the value for $f(x, y)$. In most cases score values are numbers. Therefore several predefined classes for the score are already provided by *eaLib*. There is an interface called *Score* in package *mss.ea.core* which defines some useful methods for accessing a score. These are the methods *isBetter(Score s), isEqual(Score s), isWorse(Score s)* and *value()*. The first three are used for comparing scores to each other while the last one is used for return the underlying score object.

In package *mss.ea.eval* the user can find everything he needs for evaluating individuals. For to mention is the abstract base class *AbstractScore* which gives a default implementation of the score interface. This class implements the java interface *Comparable*, so the method *compareTo(Object o)* must be implemented in all subclasses.

Four predefined subclasses for scores already exist: *DoubleScore* , *FloatScore* , *IntegerScore* and *LongScore* , holding appropriate primitive data types. For most applications these score definitions will do their job, but as shown later, the user is able to define scores for special tasks as well.

5.2 Comparing Scores

How do we compare scores ? Have a look at the following piece of code:

```
FloatScore f1 = new FloatScore ( ( float ) 2.5 );
FloatScore f2 = new FloatScore ( ( float ) 10.5 );

System.out.println ( "comparison 1: " + f1.compareTo( f2 ) );
System.out.println ( "comparison 2: " + f2.compareTo( f1 ) );
```

Listing 5.1: Score Comparison I

This would be the output:

```
-> comparison 1 : -1
-> comparison 2 : 1
```

In this piece of code it can be seen, that there is a default comparison mechanism already built-in. The score *f2* seems to be better than the score *f1*, that means a higher score is better. This may be useful in some application cases, but in our example from the last chapter. In this example we are looking for the minimal target objective value. Of course this default behaviour can be changed – by use of so called comparators. There is an interface in package *mss.ea.core* called *ScoreComparator* . The implementation of this interface can be used to changed the behaviour of such comparisons.

In package *mss.ea.eval* some default implementations of the interface can be found. The class *ReverseScore-Comparator* fits our needs. We can changed the code from above to the following:

```
FloatScore f1 = new FloatScore ( ( float ) 2.5 );
FloatScore f2 = new FloatScore ( ( float ) 10.5 );
ReverseScoreComparator comp = new ReverseScoreComparator();

System.out. println ( "comparison␣1␣:␣" + comp.compare( f1, f2 ) );
System.out. println ( "comparison␣1␣:␣" + comp.compare( f2, f1 ) );
```

Listing 5.2: Score Comparison II

The output look like this:

```
-> comparison 1 : 1
-> comparison 2 : -1
```

Now we have the desired behaviour: a lower target objective value is now better than a higher one. The comparator can be used to compare scores (and individuals, see next section) correctly for our example.

5.3 Score Assignment and Individual Comparison

Of course the evaluated score must be assigned to an individual. This is quite a simple task:

```
Individual i = new Individual ();
FloatScore f = new FloatScore ( ( float ) 10.0 );
i. setScore ( f );
```

Listing 5.3: Assigning a Score to an Individual

The class *Indivdidual* also implements the *Comparable*-interface. So that task classes implementing the interface *IndividualComparator* are used. A default implementation is available in package *mss.ea.eval* and is called *DefaultIndividualComparator* . This class uses internally a default score comparator for comparing individuals by looking at their scores. The desired behaviour for our example can be reached by setting another score comparator. This is shown in the next listing:

```
Individual i1 = new Individual ();
Individual i2 = new Individual ();

i1. setScore ( new FloatScore ( ( float ) 2.5 ) );
i2. setScore ( new FloatScore ( ( float ) 10.5 ) );

ReverseScoreComparator sComp = new ReverseScoreComparator();
DefaultIndividualComparator iComp = new DefaultIndivdualComparator( comp );

System.out. println ( "comparison␣1␣:␣" + iComp.compare( i1, i2 ) );
System.out. println ( "comparison␣1␣:␣" + iComp.compare( i2, i1 ) );
```

Listing 5.4: Comparing Individuals by Use of Individual Comparators

This produces the desired output:

```
-> comparison 1 : 1
-> comparison 2 : -1
```

5.4 Creation of User-Defined Scores

There are application cases in which only one target objective value can't be used. A multicriterion optimization problem is an example for that. So a user could require a special score definition. This can also be done with *eaLib*. This section shows how an own score class can be defined.

As an example we could require a score that is composed of two integer values. Each of them should be as small as possible. So we first define a new class which holds the two values together. The class includes a constructor and access methhods to the instance variables.

```
public class DoubleInteger {
  int first , second;

  public DoubleInteger ( int f , int s ) {
    first = f;
    second = s;
  }

  public int getFirst () {
    return first ;
  }

  public int getSecond () {
    return second;
  }

}
```

Listing 5.5: Class Holding Two Integer Variables

Now we can create a new score class:

```
public class DoubleIntegerScore extends AbstractScore {
  DoubleIntegerScore score;

  public DoubleIntegerScore ( DoubleInteger di ) {
    score = di;
  }

  public Object value () {
    return score;
  }

  public int compareTo( Object o ) {
    DoubleIntegerScore other = ( DoubleIntegerScore ) o;
    if ( first < other . getFirst () && second < other . getSecond () ) {
      return -1;
    }
    if ( first > other . getFirst () && second > other . getSecond () ) {
      return 1;
    }
    if ( first < other . getFirst () && second > other . getSecond () ) {
```

```
        return 0;
22    }
      if ( first > other.getFirst() && second < other.getSecond() ) {
24      return 0;
      }
26  }

28 }
```

Listing 5.6: Score Class with DoubleInteger

With the above code the score class is complete. The comparison of two scores is done in a way that is usual for multicriterion problems. A score is better than another if both parts are smaller, it is worse if both parts are bigger and is equal to the other score if only one part is smaller.

5.5 Evaluation of Individuals

The last section of this chapter deals with the evaluation of individuals itself. The calculation of a valid score is always problem-specific. Therefore it must be always implemented by the user. There can be only little help by the toolkit for the support of this task. In *eaLib* the evaluation is done by classes derived from the abstract base class *ScoreEvaluation* in package *mss.ea.eval*. This class extends the base class *GeneticOperator* and implements the interface *StreamProcessor* , but the explaination of these classes will be left aside for the moment. If we look at the API, we find out that *ScoreEvaluation* defines one abstract method: *evaluate(Individual i)*. In this method the calculation of the score is executed. If we want to implement the calculation of the score for our sphere example, we have to derive a new class from this abstract base class and fill out the evaluate method:

```
public class SphereEvaluation extends ScoreEvaluation {
2
        public Score evaluate( Individual ind ) throws ScoreEvaluationException {
4           try {
                ChromosomeSet set = ind.getChromosomeSet();
6               float f0 = ((FloatChromosome) set.get(0)).floatValue();
                float f1 = ((FloatChromosome) set.get(1)).floatValue();
8               float scoreValue = f0 * f0 + f1 * f1;
                FloatScore score = new FloatScore( score );
10              ind.setScore( score );
                return score;
12          } catch ( Exception e ) {
                throw new ScoreEvaluationException( "score evaluation failed" );
14          }
        }
16
}
```

Listing 5.7: Defining a Score Evaluation Class

The calculation works as follows: first the chromosome set from the individual is acquired. We know that our individuals contain two chromosomes of type *FloatChromosome* , so we can extract the float values from them. After that the score value is calculated and a new instance of a score class is created from this value. The instance is assigned to the individual and is returned from the method.

We have created an evaluated individual with this piece of code. Note the try-catch block: this block throw an exception as indicator if the calculation of the score fails.

Chapter 6

Fitness Scaling

We already explained the target objective value in the last chapter, now we continue with the fitness of individuals. The fitness value is normally defined as a mapping of the score to a non-negative range of values. Therefore the fitness is defined as a non-negative float value. The interface *Fitness* and the default implementation *DefaultFitness* are already available for use. The use, the assignment and the creation of self-defined fitness classes works much in the same way as for scores. For that reason they are not explained in more detail here.

The calculation of the fitness value is done in subclasses of the abstract base class *FitnessScaling*. The calculation mechanism of much the same as in *ScoreEvaluation*, therefore we do explain it in detail once again. But there is a difference to the score evaluation: several approaches for fitness assignment already exist. Some of these methods are already implemented in *eaLib* and will be explained in the next sections.

In the following equations the fitness is indicated by F and the score by$S$.

6.1 Linear Scaling

Linear scaling is an approach that maps the score directly to the fitness. The fitness is calculated by the following equation:

$$F = a \cdot S + b \tag{6.1}$$

The values for a and b can be specified in the constructor.

6.2 ReciprocalScaling

Reciprocal scaling is also a direct mapping:

$$F = \frac{a}{S + b} + c \tag{6.2}$$

The values for a, b and c can be specified in the constructor.

6.3 Logarithmic Scaling

Logarithmic Scaling is also a direct mapping:

$$F = a - \log S \tag{6.3}$$

The value for a can be specified in the constructor.

6.4 Exponential Scaling

Exponential Scaling is also a direct mapping:

$$F = (a \cdot S + b)^k \tag{6.4}$$

The values for a, b and k can be specified in the constructor.

6.5 Linear Ranking Scaling

to be written . . .

6.6 Non-linear Ranking Scaling

to be written . . .

Chapter 7

Selection Mechanisms

To be written . . .

Chapter 8

Recombination of Individuals

All recombination operators are contained in the package *mss.ea.rec*. The toplevel class is the abstract base class *Recombination*, which does not not provide any functionality at this moment. This class has two subclasses, *IndividualRecombination* and *ChromosomeRecobination*. The first one is the class responsible for recombining whole individuals. The other one is an abstract base class for various recombination operators working with different data types.

8.1 Recombining Bitvectors

Section still missing.

8.2 Recombining Real or Integer Valued Variables

Section still missing.

8.3 Recombining Arrays

8.3.1 Multipoint Array Recombination

This is an implementation of the well-known multipoint recombination. This operator swaps sections of arrays between selected points. Parameters to this operator are:

- *intersectionNumber* – the number of intersection points for the recombination. The selection of the points itself is random-based.
- *intersectionPoints* – an integer array with the intersection points.

Examples for multipoint crossover is shown in figure **??**. The left side shows the recombination with the *intersectionNumber* set to 1, in the right part the intersection points were set to $[2, 7]$.

8.4 Recombining Lists

Section still missing.

8.5 Recombining Strings

Section still missing.

Chapter 9

Mutation of Individuals

All mutation operators are contained in the package *mss.ea.mut*. The toplevel class is the abstract base class *Mutation*, which does not not provide any functionality at this moment. This class has two subclasses, *IndividualMutation* and *ChromosomeMutation*. The first one is the class responsible for mutating whole individuals. The other one is an abstract base class for various mutation operators working with different data types.

9.1 Mutating Bitvectors

Section still missing.

9.2 Mutating Real or Integer Valued Variables

Mutation of real or integer valued variables is done using the same scheme for the four supported data types Integer, Long, Float and Double. Each data type provides the same mutation mechanisms in it's particular range of values. Representative for all data types only the mutation for float numbers are explained in the following sections.

9.2.1 FloatStepMutation

The FloatStepMutation (FSM) is a simple mutation of float numbers. The mutation is done by adding a random float number f_r to the original float number f_c representing the chromosome. The class requires the user set a bound b for f_r. The generated random number is in the range between the bound and 0.0. So the mutation can be written as:

$$f_c = f_c + f_r \quad \text{with} \quad f_r \in \begin{cases} [0, b] & \forall \quad b > 0, \\ [b, 0] & \forall \quad b < 0. \end{cases} \tag{9.1}$$

The value of b is set to a default value of 1.0 in case that b is not given in the constructor. The default mutation propability is set to 0.05, if not given.

9.2.2 FloatRangeMutation

The FloatRangeMutation (FRM) works in a similiar way to FSM. The mutation is also done by adding a random float number f_r to the original float number f_c. The difference is, that f_r is in a range, which must be specified by a lower and an upper bound (b_l and b_u). Mathematically the mutation can be expressed like this:

$$f_c = f_c + f_r \quad \text{with} \quad b_l < f_r < b_u \tag{9.2}$$

The default values for b_l and b_u are -1.0 and 1.0, which are also used in case that $b_u < b_l$. The default mutation propability is 0.05.

9.2.3 FloatRelativeRangeMutation

The FloatRelativeRangeMutation (FRRM) works the same way as FRM with the difference that the mutation range is given as a percentage p. So the mutation can be expressed in a similiar way as FRM:

$$f_c = f_c + f_r \quad \text{with} \quad -f_c \cdot p < f_r < f_c \cdot p \tag{9.3}$$

The default value for p is 0.1, which is also used in case of a given negative value. The default mutation propability is 0.05.

9.3 Mutating Arrays

9.3.1 ReverseArrayMutation

This operator reverses the sequence of the objects in the array. Optionally a reversal of a part of the array is possible. Parameters to this operator are:

- *propability* – the propability that the mutation takes place.
- *rangePercentage* – the number of elements to be reversed. Valid values are between 0.0 and 1.0. Example: a value of 0.5 on an array with 10 elements selects a sequence of 5 elements and reverses these. The other 5 elements stay untouched.
- *lower bound/upper bound* – the lower and upper bound of the array. Elements between the bounds are reversed, all other elements stay untouched.

The mechanism is shown in figure 9.1. The left side shows the mutation of the whole array, in the right part the value of *rangePercentage* is set to 0.5.

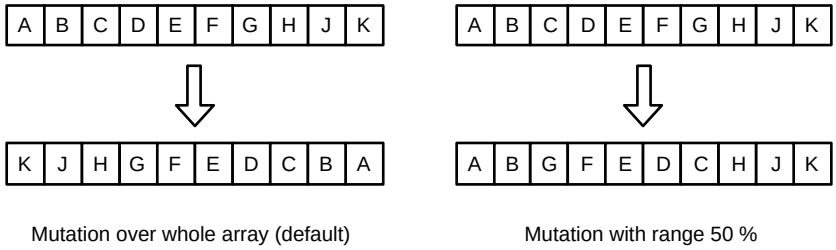

Figure 9.1: Reverse Array Mutation.

9.3.2 RotateArrayMutation

This operator rotates the contents of an array. Parameters to this operator are:

- *propability* – the propability that the mutation takes place.
- *percentage* – the number of single rotations that are performed during the mutation. Valid values are between 0.0 and 1.0.
- *number* – the number of single rotations that are performed during the mutation.

The mechanism is shown in figure 9.2. The left side shows the mutation with the value of *number* set to 1, in the right part the value of *percentage* is set to 0.5.

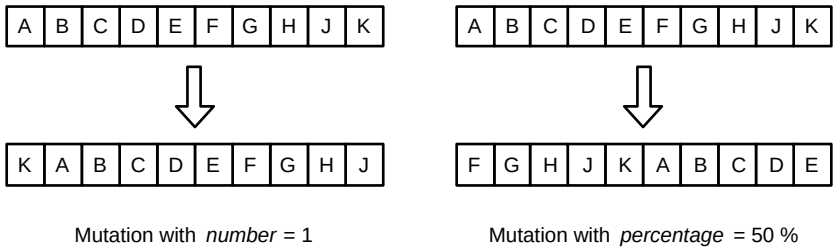

Figure 9.2: Rotate Array Mutation.

9.3.3 ScrambleArrayMutation

This operator scrambles the whole array oor just a part of it. Parameters to this operator are:

- *propability* – the propability that the mutation takes place.
- *percentage* – percentage value of the number of elements to be scrambled. Valid values are between 0.0 and 1.0.
- *number* – the number of elements to be scrambled.

An example is shown in figure 9.3. The positions of randomly selected elements from the array are scrambled.

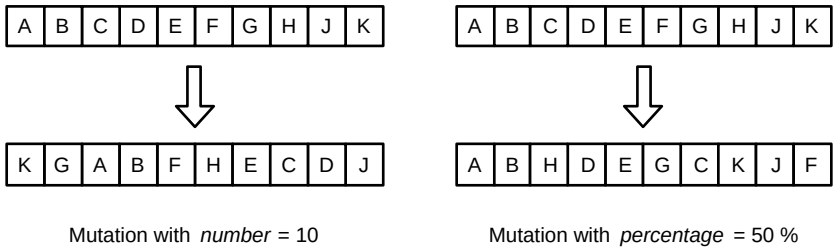

Figure 9.3: Scramble Array Mutation.

9.3.4 ShiftArrayMutation

This operator will cut out random elements, shift the remaining elements upwards and append the elements, that were clipped, to the end of the array. Parameters to this operator are:

- *propability* – the propability that the mutation takes place.
- *percentage* – percentage value of the number of elements to be cut out. Valid values are between 0.0 and 1.0.
- *number* – the number of elements to be cut out.

An example is shown in figure 9.4. In the left part two elements (’E’ and ’H’) were cut out. In the right part 50% of the elements were cut out (5 elements: ’C’, ’D’, ’F’, ’H’ and ’J’).

9.3.5 SubstitutionArrayMutation

This operator will replace randomly selected elements with elements selected randomly from a given substitution list. Parameters to this operator are:

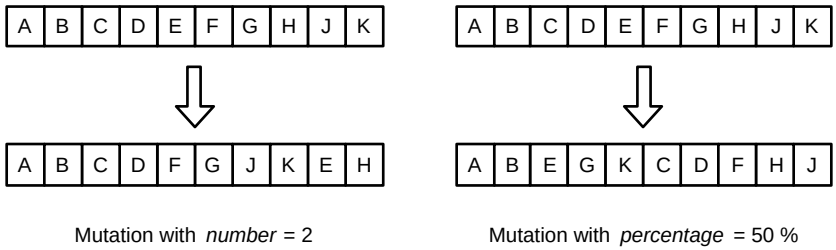

Figure 9.4: Shift Array Mutation.

- *propability* – the propability that the mutation takes place.
- *percentage* – percentage value of the number of elements to be replaced. Valid values are between 0.0 and 1.0.
- *number* – the number of elements to be replaced.

An example is shown in figure 9.5. The substitution pool is shown on the right side. Five elements of the array were cut out and replaced by randomly selected elements from the substitution pool.

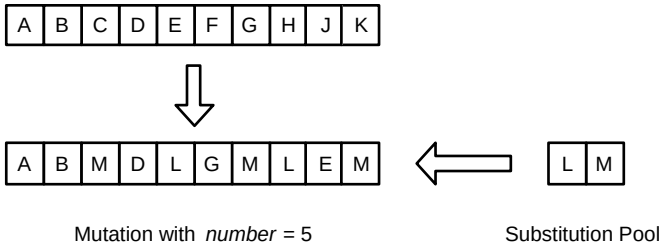

Figure 9.5: Substitution Array Mutation.

9.3.6 SwapArrayMutation

This operator swaps two randomly selected elements in the array. Parameters to this operator are:

- *propability* – the propability that the mutation takes place.

An example is shown in figure 9.6. The positions of two randomly selected elements of the array are swapped.

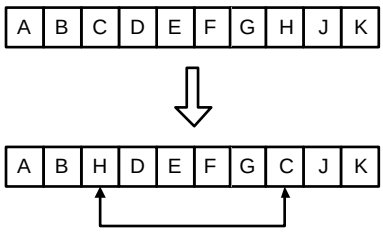

Figure 9.6: Swap Array Mutation.

9.4 Mutating Lists

9.4.1 ReverseListMutation

This operator works in the same way as ReverseArrayMutation. See the appropriate section for more information.

9.4.2 ScrambleListMutation

This operator works in the same way as ScrambleArrayMutation. See the appropriate section for more information.

9.4.3 SwapListMutation

This operator works in the same way as SwapArrayMutation. See the appropriate section for more information.

9.5 Mutating Strings

Section still missing.

Chapter 10

Algorithm Creation

This chapter gives an overview over the creation of algorithms.

10.1 Individual Streams

First of all we will introduce the term *individual stream*. Most of the operators of the library rely on such streams, so their understanding is essential.

Almost all genetic operators are working on a collection of individuals. For that reason an abstract data structure has been defined which is specified by the interface *IndividualStream*. A stream is nothing else than a collection of individuals. It provides methods for adding and removing individuals as well as for iterating over the stream. A stream contains an internal pointer, so insertion at the end of the stream or at the position of the pointer is possible. The pointer is also used for iteration, as shown in the next listing:

```
public void printStream ( IndividualStream stream ) {
  for ( stream.reset (); stream.hasNext (); ) {
        Individual ind = stream.next ();
              System.out.println ( ind );
        }
}
```

Listing 10.1: Iteration over an individual stream

Individual stream classes are specified by the interface *IndividualStream*. This interface defines several methods for putting individuals on the stream, removing them from the stream and iterating over all individuals contained in the stream. The underlying data structures are hidden from the user. There are three default implementations of the interface: *ArrayStream, ListStream* and *VectorStream* with the appropriate underlying data structures *ArrayList, LinkedList* and *Vector*. For most application cases the *ArrayStream* is useful because it provides the highest performance.

Note : It is likely, that the interface *IndividualStream* will be replaced in the future by the interface *IndividualCollection* which is derived from the Collection interface in the java.util package. Changes in most of the classes will be necessary (this means a lot of work), so the use of individual stream will be OK for foreseeable future.

10.2 Algorithm Components

All algorithms (independent from their type) can be build from independent operators. These operators produce and consume data or process data in a particular way. Every algorithm component can be assigned a particular category. This section introduces these categories. A user of *eaLib* normally does not get in touch with the communication of these components, but he should know about their sense to be able to create algorithms.

10.2.1 Sources

A source is an algorithm component which produces data. Every source has exactly one output and no input.

10.2.2 Sinks

A sink is able to store data. That's why sinks have exactly one input and no output.

10.2.3 Connectors

Connectors transform data in some way. They have exactly one input and exactly one output.

10.2.4 Forks

Forks are able to split up data. They have exactly one input and an arbitrary number of outputs. Forks can be differentiated into routers and multicasters. Routers route the incoming data from the input to one of the outputs. In contrast multicasters send the incoming data to all of the outputs.

10.2.5 Mergers

Mergers are the opposite of forks. They have an arbitrary number of inputs and excatly one output. They can be differentiated into collectors and combiners. Collectors react on every incoming data from an input. Combiners wait for every input to deliver data.

10.2.6 Conduits

Conduits are the general form of algorithm components. They can have an arbitrary number of inputs and outputs.

10.3 Turning Genetic Operators into Algorithm Components

This section explains how genetic operators are turned into algorithm components. Every genetic operator can be assigned to one of the components introduced in the last section.

10.3.1 Initialization

Let's start with the initialization. The initialization of a population in a genetic algorithm has the task to create (mostly random) individuals. The form of the individuals are problem-specific, so in most cases the user is forced to implement his own initialization operator.

Imagine individuals which represent a solution of a mathematical function in three dimensions. For that example an individual contains two chromosomes of type float. An appropriate operator must produce valid individuals containing these chromosomes. The following listing shows an operator providing this task.

```
public class MyInitialization extends Initialization {
    public MyInitialization ( int numberOfIndividuals ) {
        super( numberOfIndividuals );
    }
    public ChromosomeSet createSet() {
        ChromosomeSet set = new ChromosomeSet();
        FloatChromosome f0 = new FloatChromosome( RandomUtil.randomFloat( -10.0, 10.0 ) );
        FloatChromosome f1 = new FloatChromosome( RandomUtil.randomFloat( -10.0, 10.0 ) );
        set.add( f0 );
        set.add( f1 );
        return set;
```

```
12          }
}
```

Listing 10.2: Creating an Intialization Operator

There is an abstract base class for the initialization task called *Initialization* in package *mss.ea.ini*. We derive our own class from this abstract base class (line 1).

Chapter 11

Debugging Facilities

To be written . . .

Chapter 12

A Look into the Future

This chapter provides a look into the magical glass sphere to figure out what features will be or could be implemented in future versions of *eaLib*. The current version is in an experimental state, so there are probably many bugs lurking out there. The task of fixing issues has got the highest priority at the moment. Furthermore some more examples should be implemented to show various features of the library. The next few lines contain some ideas what to implement in future versions:

- Different Population Models: The common case for a population is holding all individuals in a vector. Recombination can be done between randomly chosen individuals. But there are other population models, for instance holding individuals in a matrix, allowing recombination only between neighbours. The core functionality for that already exists, so this feature could be implemented only within a short time.
- Migration between populations: it is already possible to split up and merge individual streams, so an extension in this direction should be easy to implement.
- Parallelization: The longer the calculation of a score lasts, the more performance could be achieved with disposing these calculations to different computers within a local network. To implement that, some knowledge about handling network connections is necessary. I do not have this knowledge, so this task is likely to move further backwards in the development process. Although the foundations are laid (with a working message passing system), there is currently no timeframe for that.
- Resource Sharing Model: An algorithm can be run using different populations sharing resources. This can improve the search space coverage and the convergence of the algorithm. For me this is quite an important feature.
- Graphical User Interface: By pressing the genetic operators into the scheme of Java Beans, there is the possiblility to create a GUI, that makes it possible to put algorithms together by drag and drop (or click and place or whatever . . .).

Chapter 13

License Issues

This chapter deals with the licensing conditions of *eaLib*.

13.1 The GNU General Public Licence

The following is the text of the GNU General Public Licence, under the terms of which this software is distrubuted.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

13.1.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

13.1.2 Terms and conditions for copying, distribution and modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the

distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. **Because the Program is licensed free of charge, there is no warranty for the Program, to the extent permitted by applicable law. except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Program is with you. Should the Program prove defective, you assume the cost of all necessary servicing, repair or correction.**

12. **In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the**

use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the Program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

END OF TERMS AND CONDITIONS

13.1.3 Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

13.2 eaLib commercial license

LICENSE AGREEMENT FOR COMMERCIAL APPLICATIONS

This is the preliminary non-GPL license for eaLib. This can be used for commercial applications of the software. This license will become valid in case that eaLib becomes available as a commercial product in the future.

Following are a draft of the current terms of this non-GPL license agreement. Comments are welcome - particularly those of the legal counsel of interested purchasers.

1. **Preamble**
   Please read this license agreement (called the ”Agreement”) carefully. Your use or installation of the software or any related documentation (called the ”Software”) indicates your acceptance of the following terms and conditions. If you do not agree to these terms and conditions, you may not install or use the Software.

   eaLib is dual licensed, under both the GPL and the following non-GPL license.

   The GPL license allows you to freely distribute eaLib within GPL software. This means that both source code (java files) and compiled code (class or native code) of the software must be available, and that any person you distribute it to has these same rights. Since the GPL kind of ”open source” does not suit everyone (notably for most commercial software), eaLib is also available under the following non-GPL license.

   This non-GPL license relieves you of all GPL obligations, apart from clause (7), which preserves the open source maintenance benefits of sharing bug-fixes and of peer review of enhancements.

   This non-GPL license imposes the further restriction, beyond the GPL, of clause (6), which prevents you from competing with eaLib (eg selling eaLib itself, or a competing product, or of ”forking” the project). It also restricts you to distribute eaLib only within a specifically nominated ”named product”, or named product suite.

   This non-GPL license does not allow you to give other people the right to distribute eaLib. It only gives *you* the right to distribute. This means that eaLib cannot be converted into LGPL, BSD or Apache style licenses etc via this license, or distributed under such a license.

2. **Definitions**
   The ”Software” is the eaLib software, including, but not limited to, the eaLib Library, examples and any included documentation.

   A ”commercial use” is:

   (1) the use of the Software or any work derived from the Software in connection with, for or in aid of the generation of revenue, such as in the conduct of Licensee’s daily business operations; or

   (2) any copying, distribution or modification of the Software or any work derived from the Software to any party where payment or other consideration is made in connection with such copying, distribution or modification, whether directly (as in payment for a copy of the Software) or indirectly (including but not limited to payment for some good or service related to the Software, or payment for some product or service that includes a copy of the Software ”without charge”).

3. **Ownership and License**
   The Software is owned by the Institute of Electronic Circuits and Systems (ESS), which is part of the Technical University of Ilmenau, Germany or one of its subsidiaries and is copyrighted and licensed.

4. **Warranty Disclaimer and Limitation of Liability**
   ESS licenses the Software to you on an ”as is” basis, without warranty of any kind. ESS hereby expressly disclaims all warranties or conditions, either express or implied, including, but not limited to, the implied

warranties or conditions of merchantability and fitness for a particular purpose. You are solely responsible for determining the appropriateness of using this Software and assume all risks associated with the use of this Software, including but not limited to the risks of program errors, damage to or loss of data, programs or equipment, and unavailability or interruption of operations. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you.

ESS will not be liable for any direct damages or for any special, incidental, or indirect damages or for any economic consequential damages (including lost profits or savings), even if ESS has been advised of the possibility of such damages. ESS will not be liable for the loss of, or damage to, your records or data, or any damages claimed by you based on a third party claim. Some jurisdictions do not allow for the exclusion or limitation of incidental or consequential damages, so the above limitations or exclusions may not apply to you.

5. **Distribution of eaLib**
   This license grants you a non-exclusive, non-transferable, perpetual license, to distribute the Software within a named product or product suite, in return for a one time fee.

6. **Competition with eaLib**
   This named product or product suite of yours must not compete with the Software. The Software must be used only as a minor component, embedded, in a supporting role, incidental to the primary function of your named product or product suite. The Software API (or functional equivalent) must not be exposed to the user of your product. You may not compete with the Software.

7. **Modification of eaLib**
   You have the right to modify the Software (eg: bugfixes, enhancements etc); however, any such changes must be relayed back to ESS, and split copyright in them assigned to ESS.

   Explanatory Note: ”Split copyright” means that both you and ESS acquire the same rights to the changes, so that ESS may incorporate the changes back into JSX, without mixing copyright ownership. Incorporating changes allows wide testing and peer review of your changes, and for further enchancements to be built on top of yours. You can forget about your bug fix or enhancement - you do not need to re-apply it to future releases, or to maintain it. You will be credited for your contribution, and of course, will also recieve the bug fixes etc from other licensees under this term.

Bibliography

[Ant] Ant. `http://jakarta.apache.org/ant/index.html`.

[GJS96] James Gosling, Bill Joy, and Guy L. Steele. *The Java Language Specification*. Addison Wesley, Paris, Reading, 1996.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, Massachussetts, 1989.

[JDK] Java™2 sdk, standard edition 1.3. `http://java.sun.com/j2se/1.3`.

[JSX] Jsx java serialization to xml. `http://www.csse.monash.edu.au/~bren/JSX`.

[JUn] Junit. `http://www.junit.org`.

[Log] Log4j. `http://jakarta.apache.org/log4j/docs/index.html`.

[Rec73] Ingo Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog, Stuttgart, 1973.

[Xer] Xerxes java parser. `http://xml.apache.org/xerces-j/index.html`.

Index