Evolutionary Algorithm Sandbox:
A Flex-Based Graphical User Interface for Evolutionary Algorithms

EEC693: Population-Based Optimization - Project Report

Brent Gardner
Department of Electrical and Computer Engineering
Cleveland State University
Cleveland, OH 44115

brent@ebrent.net

December 4, 2008

Table of Contents

Abstract

The Evolutionary Algorithm (EA) Sandbox is an Adobe Flex-based graphical user interface (GUI) that allows for a visual demonstration of evolutionary algorithm simulations. It allows the user to select many common evolutionary parameters and algorithms (such as a basic genetic algorithm and biogeography-based optimization), run a simulation, and view the results after each evolution. The EA Sandbox is meant to be a learning tool and starting point for users, giving them the ability to examine how different parameters and algorithms perform for a number of common benchmark functions. The EA Sandbox can also be easily extended to incorporate more algorithms and problem functions.

1 Introduction

A graphical user interface (GUI) for evolutionary algorithms (EAs) can help users learn quickly the effects of different parameters and algorithms have on solving certain problems. One of the most popular GUI for EAs is included in a MATLAB toolbox[1]. However, one must have both MATLAB and the toolbox to experiment with the GUI. There are few other examples of GUIs for EAs[2], especially ones that are web-based and easily accessible. Using the quasi-open source language Flex[3] from Adobe, a web-based GUI for EAs was developed. It includes many of the features of the MATLAB GUI, and is accessible from any web browser with Adobe Flash Player plug-in installed. This EA "sandbox" enables anyone to learn about different EAs and how their parameters affect performance and results, without the need for expensive computational software. The GUI can also be used by instructors to show students how different EAs perform in a visually stimulating manner.

The Flex development environment is split into two areas: MXML, which provides the graphical interface, and ActionScript, which provides background processes. As seen in Fig. 1, Flex is very similar to HTML and JavaScript, but is compiled into a Flash file. Interactive GUIs are much easier to write and manipulate using Flex and Flash than HTML and JavaScript. HTML and JavaScript are also limited to one low-bandwidth communication method, Ajax.

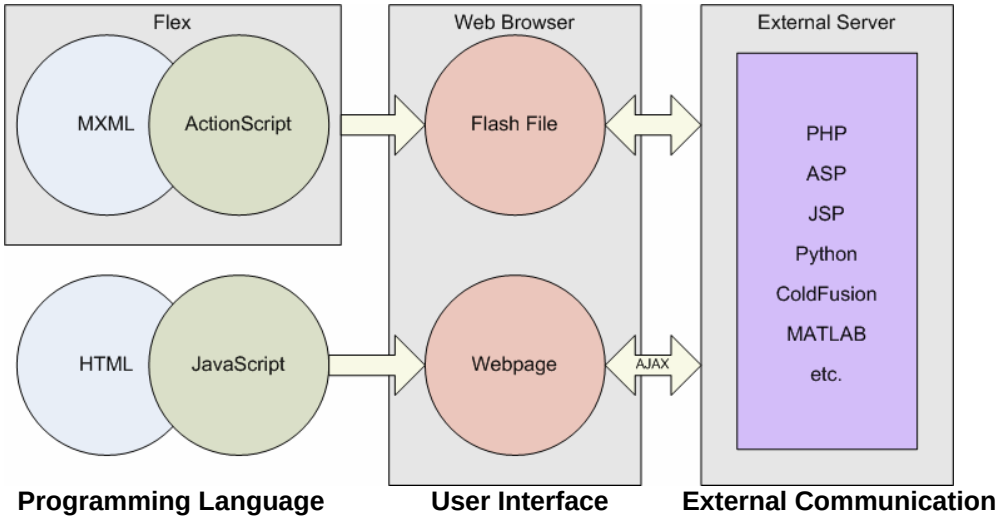

Figure 1 – Comparison diagram between Flex and HTML/JavaScript

2 Design

2.1 Graphical Layout (MXML)

The layout of the GUI is broken into three panes, as seen in Figure 2: a fitness graph, an individuals table, and a control panel that provides for algorithm configuration. The control panel includes the stop conditions, population properties, problem parameters (including fitness functions[4]), and common options. The control panel also includes a tabbed-menu of algorithms: Basic GA (Genetic Algorithm), BBO (Biogeography-Based Optimization), and More (reserved for extendibility) tabs. Parameters associated with each of these algorithms are listed in their corresponding tabs and configurable by the user. When a user has selected each of these settings, the "Start" button is pressed and the simulation executes in ActionScript. Live feedback of the results is displayed in the fitness graph and individuals table. Fitness graphs can be compared between runs using different algorithms and parameters.

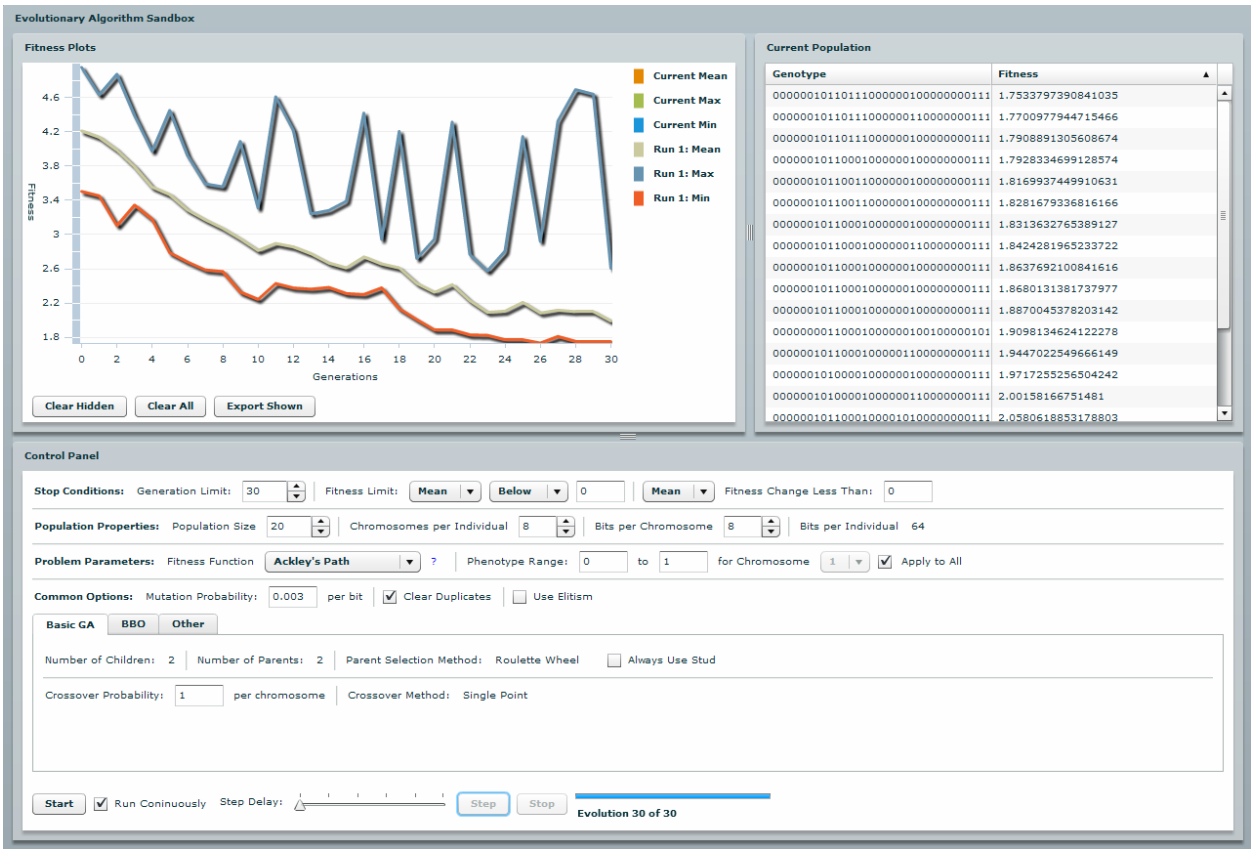

Figure 2 – Graphical layout of the EA Sandbox

The EA Sandbox panes, or tiles, can also be viewed as tabs for low-resolution displays. A tabbed view will make each pane fill the browser window. This view may also be appropriate if the control panel is extended with enough feature to warrant a larger display area. The method of viewing (tiled or tabbed) is asked upon loading and can only be changed by reloading the application.

2.2 Background Processes (ActionScript)

There are two main functions provided by ActionScript: display control and simulation execution. As the user selects different options in the control panel, other options are updated to reflect changes (such as Bits per Individual or the BBO probabilities graph). ActionScript also updates the fitness graph and the population table with data from the simulation. A flowchart of the display control is shown in Figure 3, with details of simulation execution shown in Figure 4.

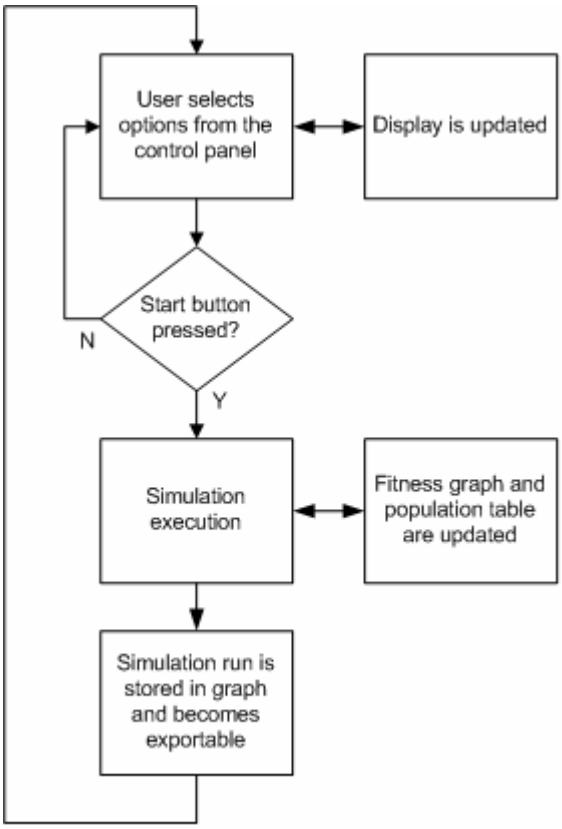

Figure 3 – Display control flowchart

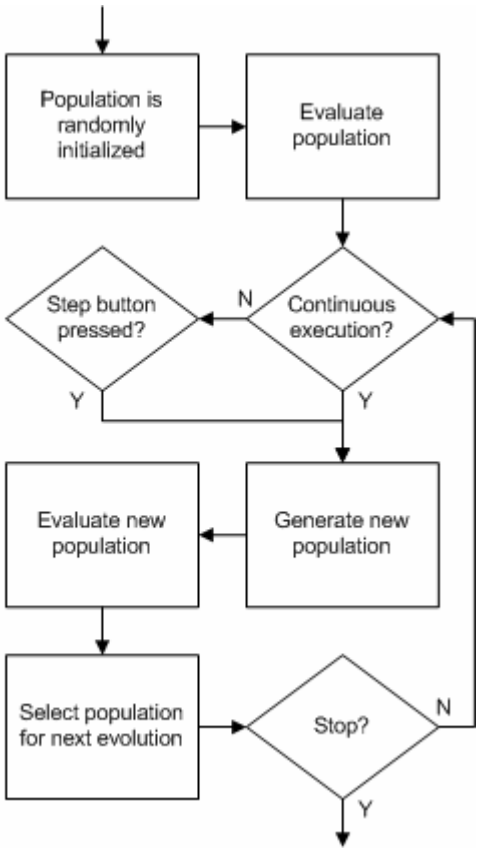

Figure 4 – Simulation execution flowchart

The population evaluation and selection will vary depending on the options and algorithm selected. The evaluation involves the fitness function and phenotype ranges to determine the fitness, or Habitat Suitability Index (HSI). The selection for the next generation is based on the "Clear Duplicates" option, which will replace duplicate individuals with a random individual, and the "Use Elitism" option, which will select the most fit individuals from both old and new populations (otherwise, select only the new population). For the case of the Basic GA, two parents will be selected by roulette wheel (with a stud option), probabilistically crossed-over at a single random point, and then probabilistically mutated to create two new individuals. For the case of BBO, a new island will be created using the immigration and emigration functions, and then be probabilistically mutated.

3 Usage

Usage of the EA Sandbox is meant to be very simple and self-explanatory. The interface provides many tool-tips containing explanations which pop-up during a mouse-over of certain items. The sections below explain all of the options and features of the EA Sandbox in detail.

3.1 Control Panel

3.1.1 Stop Conditions

The stop conditions include evolution (or generation/migration), fitness, and fitness change limits. The evolution limit is the maximum number of evolutions that will be executed before the simulation stops. The fitness limit will stop the simulation if the selected fitness (min/mean/max) is above or below the given value. The fitness change limit will stop the simulation if the selected fitness changes less than the given value from one evolution to the next. Again, the simulation will stop if any of these stop conditions are met.

3.1.2 Population Properties

The population properties include the population size (or islands), number of chromosomes (or SIVs), bits per chromosome (or per SIV), and bits per individual (or island). Since ActionScript runs on the client machine, there are limits imposed on these values to prevent system instability. The population size is limited to 100 individuals, and the number of chromosomes and bits per chromosome are limited to 16. The bits per individual are calculated automatically.

3.1.3 Problem Parameters

The problem parameters include the fitness (or habitat suitability) function, and phenotype ranges for each chromosome (or for each SIV). The fitness functions included in the EA Sandbox are common algorithm benchmark functions included in a popular Matlab toolbox[4]. The phenotype ranges are the numeric values to be associated with the binary genotypes of each chromosome. These can be set differently for each chromosome, or for all by checking the "Apply to All" option.

3.1.4 Common Options

The common options include the mutation probability per bit, the clear duplicates option, and the elitism option. Mutation will occur with the probability given for every algorithm that includes the mutation function (both the Basic GA and BBO do). If the clear duplicates option is checked, duplicate individuals (or islands) will be removed from the population and replaced with random ones. If the elitism option is checked, the population used in the next generation will be selected from the most fit of both old and new populations, otherwise only the new.

3.1.5 Algorithm Selection & Configuration

An algorithm is selected by clicking on the algorithm's tab. The Basic GA algorithm is selected by default. The options for the Basic GA include always using the stud and the probability of crossover per chromosome. If the "Always Use Stud" option is checked, the most fit individual will always be one of the parents. A crossover probability of 1 guarantees crossover, as the single crossover point will not occur at either end of the chromosome. The options for the BBO algorithm include selection of immigration and emigration functions. A graph of these functions can be seen in the BBO tab. Note that the roulette wheel plots will not be valid until a simulation is started.

3.1.6 Simulation Control

Simulation control options are located below the algorithms. To start a simulation, click the "Start" button. Simulations can be run continuously or stepped, depending on the status of the "Run Continuously" option. If run continuously, the step delay can be adjusted using the slider bar from zero to five seconds. If stepped, clicking the "Step" button will execute the next evolution. The "Stop" button can be clicked at any time to end the simulation.

3.2 Fitness Graph

The fitness graph contains the plots of the mean, maximum, and minimum fitness versus evolution for each simulation run. The current run is displayed first in the legend, and completed runs are displayed below the current run. By clicking on a plot's legend item, the user can hide or show the plot on the graph. The graph will scale its fitness axis after each show or hide, allowing the user to zoom in on a specific plot. Data points can also be seen by mousing-over points on the plots. Clicking the "Clear Hidden" button will remove the hidden plots from the graph permanently. Clicking the "Clear All" button will remove all the plots from the graph permanently. Clicking the "Export Shown" button will open a window with comma-separated values for each the completed simulation plots shown. This data may be copied into a text editor to be saved. Click the "X" to close the window.

3.3 Current Population Table

The current population table lists all the individuals (or islands) in the current evolution, showing their concatenated genotype bit string and corresponding fitness. To view the genotype and phenotype for each chromosome (or SIV) of an individual, double-click on the listing. An "Individual Genotype Breakdown" window will appear with a drop-down box allowing the user to select the specific chromosome to view. Click the "X" to close the window.

4 Extendibility

The EA Sandbox not only allows users to compare existing parameters, algorithms, and problem functions, but can be extended to include other parameters, algorithms, and problem functions. By right-clicking anywhere in the EA Sandbox and choosing "View Source", users can download the source code and modify as they see fit. Brief tutorials on how to add a problem function and create a new algorithm are included in this section. A recommended development environment is Adobe Flex Builder Pro, available for free to students and faculty of educational institutions[5]. The Pro version is needed to take advantage of the graphing component.

4.1 Adding a Problem Function

Adding a new problem function to the EA Sandbox involves two steps. The first is to add the function to the drop-down list. This can be accomplished in the *init()* function located in the *easandbox.as* file. Add the line *fitFuncs.push("<function name>");* to the function, where *<function name>* is the name of the new function. The name must be different than the existing functions.

Next, add the function to the *evalFitness()* function also located in the *easandbox.as* file. The mathematical expression for the function should be inserted as a *case* statement in the *switch* code block. Most of the existing fitness functions involve looping through each chromosome (*pop[i].chroms[j]*) of each individual (*pop[i]*). The fitness is then assigned by adding the line *pop[i].fitness = <ProblemFunctionValue>;*, where *<ProblemFunctionValue>* is the calculated value of the problem function. Add the line maxFitFunc = <Boolean>, where <Boolean> is *true* (function is to be maximized) or *false* (function is to be minimized).

4.2 Adding an Algorithm

Adding a new algorithm to the EA Sandbox also involves two steps. The first is to add the algorithm to the *TabNavigator* in the *easandbox.mxml* file. Follow the format of the other algorithms and input options of the control panel to add the parameter controls of the new algorithm. Each control will need a unique *id* that will be used to retrieve its value. The new algorithm will be listed in the order placed in the *TabNavigator* code block. The *tabChange* function in the *easandbox.as* file will execute each time a tab is changed. This function can be used to change control panel parameter nomenclature, etc. if desired.

Next, the algorithm needs to be added to the *stepEA* function in the *easandbox.as* file as a *case* in the *switch* code block. The *case* value must be the same as the label of the component added to the *TabNavigator* block above. Using existing ActionScript functions or new user-created functions, create a new population (*newPop*) from the old population (*oldPop*). This algorithm will run at each evolution (or generation/migraiton).

5 Future Work

Though it is not implemented in this project, this GUI could communicate with a server to provide the simulation execution instead of ActionScript, as well as send new parameters and start new simulations. The communication could be implemented through "remoting", which is very simple to use and can interface with any web-based communication protocol (GET, POST, SOAP, WebServices, etc.). It could also be hosted on the server hosting the Flash file, and communicate directly with the simulation program. This would provide a very convenient and easy-to-use interface to run complex algorithms without having to be located by the server. An example of this would be a professor showing students the results of a EA in real-time, without having to bring the server to the classroom. The professor would simply pull up the EA Sandbox in a web browser, and have complete control of the simulation, viewing the current results as it is being executed.

6 Conclusion

The EA Sandbox was created at a visual learning tool for evolutionary algorithms. A basic set of parameters, problem functions, and algorithms was included. Simulations using these different options can be run, and the results viewed live and compared to each other. Users can quickly test different options to explore their effects, as well as add their own. Testing of the EA Sandbox has not been rigorous, and therefore might contain software bugs. Any questions, comments, or bug reports are welcome by the author.

7 References

[1] The MathWorks, Inc., "Optimization Toolbox 4.0", http://www.mathworks.com/products/optimization, 2008.

[2] Ying-Hong Liao and Chuen-Tsai Sun, "An educational genetic algorithms learning tool", IEEE Transactions on Education, Vol. 44, Issue 2, May 2001, pp. 210.

[3] Adobe Systems Inc., "Adobe Flex 3," http://www.adobe.com/products/flex, 2008.

[4] Hartmut Pohlheim, "GEATbx: Example Functions," http://www.geatbx.com/docu/fcnindex-01.html, 2006.

[5] Adobe Systems Inc., "Adobe Flex Builder 3 Pro for Education", https://freeriatools.adobe.com/flex, 2008.

Appendix A – *easandbox.mxml*

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                               layout="absolute"
                               creationComplete="init();" enabled="false"
viewSourceURL="srcview/index.html">

        <mx:Script source="easandbox.as" />

        <mx:Panel id="mainPanel" width="100%" height="100%" title="Evolutionary Algorithm
Sandbox" backgroundAlpha="0.4">
                <mx:VDividedBox id="vdivbox" width="100%" height="100%">
                        <mx:HDividedBox id="hdivbox" width="100%" height="50%">
                                <mx:Panel id="plotPanel" label="Fitness Graph" title="Fitness
Graph" width="60%" height="100%">
                                        <mx:VDividedBox id="nodivbox" width="100%" height="100%">
                                                <mx:HBox width="100%" height="100%">
                                                        <mx:LineChart id="fitnessPlot"
dataProvider="{acPlots}" width="100%" height="100%" showDataTips="true"
dataTipFunction="dataTipFunc">
                                                                <mx:verticalAxis>
                                                                        <mx:LinearAxis
id="fitnessPlotYAxis" baseAtZero="false" title="Fitness" />
                                                                </mx:verticalAxis>
                                                                <mx:horizontalAxis>
                                                                        <mx:LinearAxis
id="fitnessPlotXAxis" title="Evolutions" />
                                                                </mx:horizontalAxis>
                                                                <mx:series>
                                                                        <mx:LineSeries
displayName="Current Mean" xField="evolution" yField="meanFitness" />
                                                                        <mx:LineSeries
displayName="Current Max" xField="evolution" yField="maxFitness" />
                                                                        <mx:LineSeries
displayName="Current Min" xField="evolution" yField="minFitness" />
                                                                </mx:series>
                                                        </mx:LineChart>
                                                        <mx:Legend dataProvider="{fitnessPlot}"
itemClick="togglePlot(event);" toolTip="Click item to show/hide plot" />
                                                </mx:HBox>
                                                <mx:HBox paddingBottom="5" paddingLeft="10"
paddingRight="10">
                                                        <mx:Button label="Clear Hidden"
click="clearPlots(true);" toolTip="Clears all the hidden plots from the legend" />
                                                        <mx:Button label="Clear All"
click="clearPlots();" toolTip="Clears all the plots from the graph and legend" />
                                                        <mx:Button label="Export Shown"
click="exportShown();" toolTip="Exports the visible plots as CSV text" />
                                                </mx:HBox>
                                        </mx:VDividedBox>
                                </mx:Panel>
                                <mx:Panel id="popPanel" title="Current Population" label="Current
Population" width="40%" height="100%">
                                        <mx:DataGrid id="dgPop" dataProvider="{oldPop}"
width="100%" height="100%" doubleClickEnabled="true" itemDoubleClick="openInd(event);">
                                                <mx:columns>
                                                        <mx:DataGridColumn headerText="Genotype"
labelFunction="concatChroms"  />
                                                        <mx:DataGridColumn dataField="fitness"
headerText="Fitness" />
                                                </mx:columns>
                                        </mx:DataGrid>
                                </mx:Panel>
                        </mx:HDividedBox>
                        <mx:Panel id="ctrlPanel" title="Control Panel" label="Control Panel"
width="100%" height="50%" paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10">
                                <mx:HBox verticalAlign="middle">
                                        <mx:Label text="Stop Conditions:" fontWeight="bold"
toolTip="The simulation will stop if any of these conditions are met" />
```

```
                                        <mx:Label id="scGensLabel" text="Generation Limit:"
toolTip="The simulation will stop after this many evolutions" />
                                        <mx:NumericStepper id="scGens" minimum="1" maximum="100"
value="10" toolTip="The simulation will stop after this many evolutions" />
                                        <mx:VRule height="20" />
                                        <mx:Label id="scFitLimitLabel" text="Fitness Limit:"
toolTip="The simulation will stop if the selected fitness reaches the given limit" />
                                        <mx:ComboBox id="cbFitLimitStat" dataProvider="{new
Array('Mean','Max','Min')}" toolTip="The simulation will stop if the selected fitness reaches the
given limit" />
                                        <mx:ComboBox id="cbFitLimitLogic" dataProvider="{new
Array('Below','Above')}" toolTip="The simulation will stop if the selected fitness reaches the
given limit" />
                                        <mx:TextInput id="scFitLimit" text="0" width="50"
toolTip="The simulation will stop if the selected fitness reaches the given limit" />
                                        <mx:VRule height="20" />
                                        <mx:ComboBox id="cbFitChangeStat" dataProvider="{new
Array('Mean','Max','Min')}" toolTip="The simulation will stop if the selected fitness changes
less than the given limit" />
                                        <mx:Label id="scFitChangeLabel" text="Fitness Change Less
Than:" toolTip="The simulation will stop if the selected fitness changes less than the given
limit" />
                                        <mx:TextInput id="scFitChange" text="0" width="50"
toolTip="The simulation will stop if the selected fitness changes less than the given limit" />
                                </mx:HBox>
                                <mx:HRule width="100%" />
                                <mx:HBox verticalAlign="middle">
                                        <mx:Label text="Population Properties:" fontWeight="bold"
/>
                                        <mx:Label id="popSizeLabel" text="Population Size:" />
                                        <mx:NumericStepper id="popSize" minimum="1" maximum="100"
value="10" change="tabChange(event);" />
                                        <mx:VRule height="20" />
                                        <mx:Label id="numChromsLabel" text="Chromosomes per
Individual:" />
                                        <mx:NumericStepper id="numChroms" minimum="1" maximum="16"
value="1" change="updatePhenChrom();" />
                                        <mx:VRule height="20" />
                                        <mx:Label id="chromSizeLabel" text="Bits per Chromosome:"
/>
                                        <mx:NumericStepper id="chromSize" minimum="2" maximum="16"
value="8" />
                                        <mx:VRule height="20" />
                                        <mx:Label id="bpiLabel" text="Bits per Individual:" />
                                        <mx:Text text="{int(chromSize.value) *
int(numChroms.value)}" />
                                </mx:HBox>
                                <mx:HRule width="100%" />
                                <mx:HBox verticalAlign="middle">
                                        <mx:Label text="Problem Parameters:" fontWeight="bold" />
                                        <mx:Label id="fitFuncsLabel" text="Fitness function:" />
                                        <mx:ComboBox id="fitFunc" dataProvider="{fitFuncs}" />
                                        <mx:Label text="?" useHandCursor="true" buttonMode="true"
mouseChildren="false" click="navigateToURL(new URLRequest('http://www.geatbx.com/docu/fcnindex-
01.html'),'_blank');" toolTip="Click here to see descriptions of some of the fitness functions"
color="#0000ff" />
                                        <mx:VRule height="20" />
                                        <mx:Label text="Phenotype Range:" />
                                        <mx:TextInput id="phenFrom"
text="{cbPhenChrom.selectedItem.from}" width="50" focusOut="applyPhens();" />
                                        <mx:Text text="to" />
                                        <mx:TextInput id="phenTo"
text="{cbPhenChrom.selectedItem.to}" width="50" focusOut="applyPhens();" />
                                        <mx:Text id="phenLabel" text="for Chromosome #:" />
                                        <mx:ComboBox id="cbPhenChrom" dataProvider="{acPhens}"
labelField="chrom" enabled="{!phenAll.selected}" />
                                        <mx:CheckBox id="phenAll" selected="true"
change="applyPhens();" />
                                        <mx:Text text="Apply to All" />
                                </mx:HBox>
                                <mx:HRule width="100%" />
                                <mx:HBox verticalAlign="middle">
                                        <mx:Label text="Common Options:" fontWeight="bold" />
                                        <mx:Label text="Mutation Probability:" />
```

```
                                        <mx:TextInput id="mutProb" text="0.003" width="50" />
                                        <mx:Text text="per bit" />
                                        <mx:VRule height="20" />
                                        <mx:CheckBox id="cbClearDups" label="Clear Duplicates"
selected="true" toolTip="Removes duplicate genotypes and replaces them with random ones." />
                                        <mx:VRule height="20" />
                                        <mx:CheckBox id="useElitism" label="Use Elitism"
toolTip="Keep the best individials between both parents and children, otherwise only keep
children" />
                                </mx:HBox>
                                <mx:TabNavigator id="algSel" width="100%" height="100%"
change="tabChange(event);">
                                        <mx:VBox label="Basic GA" paddingTop="5" paddingBottom="5"
paddingLeft="10" paddingRight="10">
                                                <mx:HBox label="Evolution Parameters"
verticalAlign="middle">
                                                        <mx:Label text="Number of Children:" />
                                                        <mx:Text text="2" />
                                                        <mx:VRule height="20" />
                                                        <mx:Label text="Number of Parents:" />
                                                        <mx:Text text="2" />
                                                        <mx:VRule height="20" />
                                                        <mx:Label text="Parent Selection Method:" />
                                                        <mx:Text text="Roulette Wheel" />
                                                        <mx:Spacer width="10" />
                                                        <mx:CheckBox id="pselStud" label="Always Use
Stud" toolTip="Always use the most fit individual as a parent" />
                                                </mx:HBox>
                                                <mx:HRule width="100%" />
                                                <mx:HBox label="Crossover" verticalAlign="middle">
                                                        <mx:Label text="Crossover Probability:" />
                                                        <mx:TextInput id="xoverProb" text="1"
width="50" />
                                                        <mx:Text text="per chromosome" />
                                                        <mx:VRule height="20" />
                                                        <mx:Label text="Crossover Method:" />
                                                        <mx:Text text="Single Point" />
                                                </mx:HBox>
                                        </mx:VBox>
                                        <mx:HBox label="BBO" paddingBottom="5" paddingLeft="10"
paddingRight="10">
                                                <mx:VBox height="100%" verticalAlign="middle">
                                                        <mx:Label text="Immigration Probability
Function: " />
                                                        <mx:ComboBox id="ipFunc"
change="updateIProb(newPop);">
                                                                <mx:ArrayCollection>
                                                                        <mx:Object label="Uniform
Linear" />
                                                                        <mx:Object label="Roulette
Wheel" />
                                                                        <mx:Object
label="Exponential" />
                                                                        <mx:Object label="Sine [-
pi/2,pi/2]" />
                                                                        <mx:Object label="Hyperbolic
Tangent [-pi,pi]" />
                                                                        <mx:Object label="Step at
N/2" />
                                                                </mx:ArrayCollection>
                                                        </mx:ComboBox>
                                                        <mx:Label text="Emigration Probability
Function: " />
                                                        <mx:ComboBox id="epFunc"
change="updateEProb(newPop);">
                                                                <mx:ArrayCollection>
                                                                        <mx:Object label="Uniform
Linear" />
                                                                        <mx:Object label="Roulette
Wheel" />
                                                                        <mx:Object
label="Exponential" />
                                                                        <mx:Object label="Sine [-
pi/2,pi/2]" />
```

```
                                                        <mx:Object label="Hyperbolic
Tangent [-pi,pi]" />
                                                        <mx:Object label="Step at
N/2" />
                                                </mx:ArrayCollection>
                                        </mx:ComboBox>
                                        <mx:Spacer height="20" />
                                </mx:VBox>
                                <mx:LineChart id="probPlot" dataProvider="{acProbs}"
height="100%">
                                        <mx:verticalAxis>
                                                <mx:LinearAxis title="Probability per
SIV" minimum="0" maximum="1" />
                                        </mx:verticalAxis>
                                        <mx:horizontalAxis>
                                                <mx:LinearAxis minimum="1"
maximum="{popSize.value}" title="Fitness Rank" />
                                        </mx:horizontalAxis>
                                        <mx:series>
                                                <mx:LineSeries
displayName="Immigration" xField="rank" yField="immigration" />
                                                <mx:LineSeries
displayName="Emigration" xField="rank" yField="emigration" />
                                        </mx:series>
                                        <mx:backgroundElements>
                                                <mx:GridLines direction="both">
                                                        <mx:horizontalStroke>
                                                                <mx:Stroke weight="1"
color="#cccccc" />
                                                        </mx:horizontalStroke>
                                                        <mx:verticalStroke>
                                                                <mx:Stroke weight="1"
color="#cccccc" />
                                                        </mx:verticalStroke>
                                                </mx:GridLines>
                                        </mx:backgroundElements>
                                </mx:LineChart>
                                <mx:Legend dataProvider="{probPlot}" />
                        </mx:HBox>
                        <mx:VBox label="More . . ." paddingBottom="5"
paddingLeft="10" paddingRight="10">
                                <mx:Label text="See the User Guide to add custom
algorithms." />
                        </mx:VBox>
                </mx:TabNavigator>
                <mx:Spacer height="10" />
                <mx:HBox>
                        <mx:Button id="startBtn" label="Start" click="startEA();"
/>
                        <mx:CheckBox id="runCont" label="Run Coninuously" />
                        <mx:HBox enabled="{runCont.selected}">
                                <mx:Label text="Step Delay:" />
                                <mx:HSlider id="stepDelay" minimum="0" maximum="5"
tickInterval="1" />
                        </mx:HBox>
                        <mx:Button id="stepBtn" label="Step" click="stepEA();"
enabled="false" />
                        <mx:Button id="stopBtn" label="Stop" enabled="false"
click="stopEA();" />
                        <mx:ProgressBar id="progBar" mode="manual" maximum="100"
label="Evolution {curGen.toString()} of {scGens.value.toString()}" />
                </mx:HBox>
        </mx:Panel>
    </mx:VDividedBox>
</mx:Panel>

<mx:states>
        <mx:State name="tabs">
                <mx:RemoveChild target="{plotPanel}" />
                <mx:RemoveChild target="{popPanel}" />
                <mx:RemoveChild target="{ctrlPanel}" />
                <mx:RemoveChild target="{vdivbox}" />
                <mx:AddChild relativeTo="{mainPanel}">
                        <mx:target>
```

```
                                    <mx:TabNavigator id="tabNav" width="100%" height="100%" />
                                </mx:target>
                            </mx:AddChild>
                            <mx:AddChild relativeTo="{tabNav}" target="{plotPanel}" />
                            <mx:SetProperty target="{plotPanel}" name="title" value="" />
                            <mx:SetStyle target="{plotPanel}" name="headerHeight" value="10" />
                            <mx:AddChild relativeTo="{tabNav}" target="{popPanel}" />
                            <mx:SetProperty target="{popPanel}" name="title" value="" />
                            <mx:SetStyle target="{popPanel}" name="headerHeight" value="10" />
                            <mx:AddChild relativeTo="{tabNav}" target="{ctrlPanel}" />
                            <mx:SetProperty target="{ctrlPanel}" name="title" value="" />
                            <mx:SetStyle target="{ctrlPanel}" name="headerHeight" value="10" />
                    </mx:State>
            </mx:states>

</mx:Application>
```

Appendix B – *easandbox.as*

```
// Import needed components
import flash.events.MouseEvent;

import mx.charts.HitData;
import mx.charts.events.LegendMouseEvent;
import mx.collections.ArrayCollection;
import mx.collections.Sort;
import mx.collections.SortField;
import mx.containers.HBox;
import mx.containers.TitleWindow;
import mx.controls.Alert;
import mx.controls.Button;
import mx.controls.ComboBox;
import mx.controls.Text;
import mx.controls.TextArea;
import mx.events.CloseEvent;
import mx.events.ListEvent;
import mx.managers.PopUpManager;
import mx.utils.ObjectUtil;

// Define global variables
[Bindable]
// Probabilities for BBO
private var acProbs:ArrayCollection = new ArrayCollection();
[Bindable]
// Fitness plots for graphing
private var acPlots:ArrayCollection = new ArrayCollection();
[Bindable]
// "Old" population (previous evolution)
private var oldPop:ArrayCollection = new ArrayCollection();
[Bindable]
// "New" population (present evolution)
private var newPop:ArrayCollection = new ArrayCollection();
[Bindable]
// List of fitness (problem) functions
private var fitFuncs:Array = new Array();
[Bindable]
// Phenotype ranges for each chromosome
private var acPhens:ArrayCollection = new ArrayCollection([ { chrom: 1, from: 0, to: 1 } ] );
// Determines if fitness function is maximized
private var maxFitFunc:Boolean;
[Bindable]
// Current evolution
private var curGen:int = 0;
// Current simulation run
private var curRun:int = 0;

private function init():void
// Initializes fitness function list, BBO graph, and asks for view state
{
	nodivbox.getDividerAt(0).visible = false;

	fitFuncs.push("De Jong 1");
	fitFuncs.push("Axis parallel hyper-ellipsoid");
	fitFuncs.push("Rotated hyper-ellipsoid");
	fitFuncs.push("Moved axis parallel hyper-ellipsoid");
	fitFuncs.push("Rosenbrock's valley (De Jong 2)");
	fitFuncs.push("Rastrigin");
	fitFuncs.push("Schwefel");
	fitFuncs.push("Griewangk");
	fitFuncs.push("Sum of different power");
	fitFuncs.push("Ackley's Path");
	fitFuncs.push("Michalewicz");
	fitFunc.selectedIndex = 0;
	fitFunc.invalidateDisplayList();

	updatePhenChrom();
	for (var i:uint = 0; i < popSize.value; i++)
	{
```

```
                acProbs.addItem( { rank:i+1, immigration:(i+1)/popSize.value, emigration:1-
(i+1)/popSize.value } );
        }

        askView();
}

private function askView():void
// Creates "Select View" window and switches to tabbed view state if selected
{
        var helpWindow:TitleWindow = TitleWindow(PopUpManager.createPopUp(this, TitleWindow,
false));
        helpWindow.title="Select View";
        helpWindow.setStyle("borderAlpha", 0.9);
        var hbox1:HBox = new HBox();
        var tileBtn:Button = new Button();
        tileBtn.label = "Tiled";
        tileBtn.addEventListener(MouseEvent.CLICK,function():void {
PopUpManager.removePopUp(helpWindow); Application.application.enabled = true; });
        hbox1.addChild(tileBtn);
        var text1:Text = new Text();
        text1.text = "recommended for high display resolutions";
        hbox1.addChild(text1);
        helpWindow.addChild(hbox1);
        var hbox2:HBox = new HBox();
        var tabsBtn:Button = new Button();
        tabsBtn.label = "Tabbed";
        tabsBtn.addEventListener(MouseEvent.CLICK,function():void {
PopUpManager.removePopUp(helpWindow); currentState = "tabs"; Application.application.enabled =
true; });
        hbox2.addChild(tabsBtn);
        var text2:Text = new Text();
        text2.text = "recommended for low display resolutions";
        hbox2.addChild(text2);
        helpWindow.addChild(hbox2);
        PopUpManager.centerPopUp(helpWindow);
}

private function startEA():void
// Initializes the population and starts stepping if running continuously
{
        startBtn.enabled = false;
        stopBtn.enabled = true;
        curGen = 0;
        curRun++;
        progBar.setProgress(0,1);
        oldPop = initPop(popSize.value as int);
        evalFitness(oldPop);
        sortFitness(oldPop);
        if (algSel.selectedIndex == 1) // BBO
        {
                updateIProb(oldPop);
                updateEProb(oldPop);
        }
        if (runCont.selected)
        {
                stepEA();
        } else {
                stepBtn.enabled = true;
        }
}

private function stopEA():void
// Adds the plot of the run to the graph
{
        var newDP:ArrayCollection = new ArrayCollection();
        for (var i:uint = 0; i < acPlots.length; i++)
        {
                newDP.addItem(ObjectUtil.copy(acPlots[i]));
        }
        var tmp:Array = fitnessPlot.series;
        var meanSeries:LineSeries = new LineSeries();
        meanSeries.xField = "evolution";
        meanSeries.yField = "meanFitness";
```

```
        meanSeries.displayName = "Run " + curRun.toString() + ": Mean";
        meanSeries.dataProvider = newDP;
        var maxSeries:LineSeries = new LineSeries();
        maxSeries.xField = "evolution";
        maxSeries.yField = "maxFitness";
        maxSeries.displayName = "Run " + curRun.toString() + ": Max";
        maxSeries.dataProvider = newDP;
        var minSeries:LineSeries = new LineSeries();
        minSeries.xField = "evolution";
        minSeries.yField = "minFitness";
        minSeries.displayName = "Run " + curRun.toString() + ": Min";
        minSeries.dataProvider = newDP;
        acPlots = new ArrayCollection();
        tmp.push(meanSeries);
        tmp.push(maxSeries);
        tmp.push(minSeries);
        fitnessPlot.series = tmp;
        stopBtn.enabled = false;
        stepBtn.enabled = false;
        startBtn.enabled = true;
}

private function stepEA():void
// Executes one evolution of the simulation and checks for stop conditions
{
        if (curGen < scGens.value)
        {
                newPop = new ArrayCollection();
                var i:uint;
                var j:uint;
                var k:uint;
                switch (algSel.selectedChild.label)
                {
                        case "Basic GA":
                                for (i = 0; i < popSize.value / 2; i++)
                                {
                                        var brandNewPop:ArrayCollection =
crossOver(selectParents(oldPop));
                                        for (j = 0; j < brandNewPop.length; j++)
                                        {
                                                newPop.addItem(brandNewPop[j]);
                                        }
                                }
                                mutatePop(newPop);
                                clearDups(newPop);
                                evalFitness(newPop);
                                sortFitness(newPop);
                                break;
                        case "BBO":
                                for (i = 0; i < popSize.value; i++)
                                {
                                        newPop.addItem(ObjectUtil.copy(oldPop[i]));
                                        for (j = 0; j < numChroms.value; j++)
                                        {
                                                if (Math.random() < oldPop[i].immigration)
                                                {
                                                        var rand:Number = Math.random();
                                                        for (k = 0; k < popSize.value; k++)
                                                        {
                                                                if (rand >= oldPop[k].emigration)
                                                                {
                                                                        newPop[i].chroms[j] =
oldPop[k].chroms[j];
                                                                        break;
                                                                }
                                                        }
                                                }
                                        }
                                }
                                mutatePop(newPop);
                                clearDups(newPop);
                                evalFitness(newPop);
                                sortFitness(newPop);
                                updateIProb(newPop);
```

```
                        updateEProb(newPop);
                        break;
                }
                oldPop = newPop;
                progBar.setProgress(++curGen, scGens.value);
                if (stopConditions())
                {
                        stopEA();
                } else if (runCont.selected)
                {
                        setTimeout(function():void { callLater(stepEA) }, stepDelay.value * 1000);
                }
        } else {
                stepBtn.enabled = false;
                startBtn.enabled = true;
        }
}

private function stopConditions():Boolean
// Returns true if any stop condition is met
{
        if (curGen >= scGens.value)
        {
                return true;
        }
        var val:Number;
        switch (cbFitLimitStat.selectedItem)
        {
                case "Mean":
                        val = acPlots[acPlots.length - 1].meanFitness;
                        break;
                case "Max":
                        val = acPlots[acPlots.length - 1].maxFitness;
                        break;
                case "Min":
                        val = acPlots[acPlots.length - 1].minFitness;
                        break;
        }
        switch (cbFitLimitLogic.selectedItem)
        {
                case "Below":
                        if (val < parseFloat(scFitLimit.text))
                        {
                                return true;
                        }
                        break;
                case "Above":
                        if (val > parseFloat(scFitLimit.text))
                        {
                                return true;
                        }
                        break;
        }
        if (acPlots.length > 1)
        {
                var pval:Number;
                switch (cbFitChangeStat.selectedItem)
                {
                        case "Mean":
                                val = acPlots[acPlots.length - 1].meanFitness;
                                pval = acPlots[acPlots.length - 2].meanFitness;
                                break;
                        case "Max":
                                val = acPlots[acPlots.length - 1].maxFitness;
                                pval = acPlots[acPlots.length - 2].maxFitness;
                                break;
                        case "Min":
                                val = acPlots[acPlots.length - 1].minFitness;
                                pval = acPlots[acPlots.length - 2].minFitness;
                                break;
                }
                if (Math.abs(pval - val) < parseFloat(scFitChange.text))
                {
                        return true;
```

```
                }
        }
        return false;
}

private function calcPhen(i:uint,chrom:String):Number
// Calculates the phenotype for the given chromosome (and its number)
{
        var phen:Number = parseInt(chrom, 2);
        phen = phen / Math.pow(2,chromSize.value) * (parseFloat(acPhens[i].to) -
parseFloat(acPhens[i].from)) + parseFloat(acPhens[i].from);
        return phen;
}

private function clearDups(pop:ArrayCollection):void
// Replaces individuals of the same genotype with random individuals
{
        if (cbClearDups.selected)
        {
                for (var i:uint = 0; i < pop.length; i++)
                {
                        for (var j:uint = i + 1; j < pop.length; j++)
                        {
                                if (concatChroms(pop[i]) == concatChroms(pop[j]))
                                {
                                        pop[j] = initPop(1).getItemAt(0);
                                }
                        }
                }
        }
}

private function initPop(pSize:int):ArrayCollection
// Initializes a population with random individuals
{
        var randomPop:ArrayCollection = new ArrayCollection();
        for (var i:uint = 0; i < pSize; i++)
        {
                randomPop.addItem( { chroms: new ArrayCollection(), fitness:0, prob:0} );
                for (var j:uint = 0; j < numChroms.value; j++)
                {
                        var randomChrom:String = "";
                        for (var k:uint = 0; k < chromSize.value; k++)
                        {
                                randomChrom += Math.round(Math.random()).toString();
                        }
                        randomPop[i].chroms.addItem( { genotype:randomChrom,
phenotype:calcPhen(j,randomChrom) } );
                }
        }
        return randomPop;
}

private function selectParents(pop:ArrayCollection):ArrayCollection
// Select two parents for a mating operation
{
        var parents:ArrayCollection = new ArrayCollection();
        var numParentsNeeded:uint = 2;
        if (pselStud.selected)
        {
                parents.addItem(pop[length-1]);
                numParentsNeeded--;
        }
        var pselMethod:Object = { value:"roulette" };
        switch (pselMethod.value.toString())
        {
                case "roulette":
                for (var i:uint = 0; i < numParentsNeeded; i++)
                {
                        var rand:Number = Math.random();
                        var j:int = 0;
                        while (rand > pop[j].prob)
                        {
                                j++;
```

```
                        if (j >= pop.length)
                        {
                                j--;
                                break;
                        }
                }
                parents.addItem(pop[j]);
        }
    }
    return parents;
}

private function crossOver(parents:ArrayCollection):ArrayCollection
// Creates two children via crossover from two given parents
{
    var children:ArrayCollection = new ArrayCollection();
    children.addItem(ObjectUtil.copy(parents[0]));
    children.addItem(ObjectUtil.copy(parents[1]));
    for (var i:uint = 0; i < numChroms.value; i++)
    {
        if (Math.random() < parseFloat(xoverProb.text))
        {
            var randPoint:Number = Math.max(Math.min(Math.round(Math.random() *
chromSize.value), chromSize.value - 2), 1);
            children[0].chroms[i].genotype = parents[0].chroms[i].genotype.substr(0,
randPoint) + parents[1].chroms[i].genotype.substr(randPoint);
            children[1].chroms[i].genotype = parents[1].chroms[i].genotype.substr(0,
randPoint) + parents[0].chroms[i].genotype.substr(randPoint);
        }
    }
    return children;
}

private function mutatePop(pop:ArrayCollection):void
// Probabilistically mutates each bit in an entire population
{
    for (var i:uint = 0; i < pop.length; i++)
    {
        for (var j:uint = 0; j < numChroms.value; j++)
        {
            var newChrom:String = "";
            for (var k:uint = 0; k < chromSize.value; k++)
            {
                if (Math.random() < parseFloat(mutProb.text))
                {
                    newChrom += flipBit(pop[i].chroms[j].genotype.substr(k,
1));
                } else {
                    newChrom += pop[i].chroms[j].genotype.substr(k, 1);
                }
            }
            pop[i].chroms[j].genotype = newChrom;
            pop[i].chroms[j].phenotype = calcPhen(j,pop[i].chroms[j].genotype);
        }
    }
}

private function flipBit(bit:String):String
{
    if (bit == "1")
    {
        return "0";
    } else {
        return "1";
    }
}

private function evalFitness(pop:ArrayCollection):void
// Calculates the fitnesses for each individual using the problem function
{
    for (var i:uint = 0; i < pop.length; i++)
    {
        var j:uint;
        var k:uint;
```

```
                var sum1:Number = 0;
                var sum2:Number = 0;
                switch(fitFunc.selectedItem)
                {
                        case "De Jong 1":
                                for (j = 0; j < numChroms.value; j++)
                                {
                                        sum1 += Math.pow(pop[i].chroms[j].phenotype,2);
                                }
                                pop[i].fitness = sum1;
                                maxFitFunc = false;
                                break;
                        case "Axis parallel hyper-ellipsoid":
                                for (j = 0; j < numChroms.value; j++)
                                {
                                        sum1 += (j + 1) * Math.pow(pop[i].chroms[j].phenotype,2);
                                }
                                pop[i].fitness = sum1;
                                maxFitFunc = false;
                                break;
                        case "Rotated hyper-ellipsoid":
                                for (j = 0; j < numChroms.value; j++)
                                {
                                        sum1 = 0;
                                        for (k = 0; k <= j; k++)
                                        {
                                                sum1 += pop[i].chroms[k].phenotype;
                                        }
                                        sum2 += Math.pow(sum1, 2);
                                }
                                pop[i].fitness = sum2;
                                maxFitFunc = false;
                                break;
                        case "Moved axis parallel hyper-ellipsoid":
                                for (j = 0; j < numChroms.value; j++)
                                {
                                        sum1 += 5 * j * Math.pow(pop[i].chroms[j].phenotype, 2);
                                }
                                pop[i].fitness = sum1;
                                maxFitFunc = false;
                                break;
                        case "Rosenbrock's valley (De Jong 2)":
                                for (j = 0; j < numChroms.value-1; j++)
                                {
                                        sum1 += 100 * Math.pow(pop[i].chroms[j + 1].phenotype -
Math.pow(pop[i].chroms[j].phenotype, 2), 2) + Math.pow(1 - pop[i].chroms[j].phenotype, 2);
                                }
                                pop[i].fitness = sum1;
                                maxFitFunc = false;
                                break;
                        case "Rastrigin":
                                for (j = 0; j < numChroms.value; j++)
                                {
                                        sum1 += Math.pow(pop[i].chroms[j].phenotype, 2) - 10 *
Math.cos(2 * Math.PI * pop[i].chroms[j].phenotype);
                                }
                                pop[i].fitness = 10 * numChroms.value + sum1;
                                maxFitFunc = false;
                                break;
                        case "Schwefel":
                                for (j = 0; j < numChroms.value; j++)
                                {
                                        sum1 += -pop[i].chroms[j].phenotype *
Math.sin(Math.sqrt(Math.abs(pop[i].chroms[j].phenotype)));
                                }
                                pop[i].fitness = sum1;
                                maxFitFunc = false;
                                break;
                        case "Griewangk":
                                sum2 = 1;
                                for (j = 0; j < numChroms.value; j++)
                                {
                                        sum1 += Math.pow(pop[i].chroms[j].phenotype, 2) / 4000;
```

```
                                        sum2 *= Math.cos(pop[i].chroms[j].phenotype /
Math.sqrt(j+1));
                                }
                                pop[i].fitness = sum1 - sum2 + 1;
                                maxFitFunc = false;
                                break;
                        case "Sum of different power":
                                for (j = 0; j < numChroms.value; j++)
                                {
                                        sum1 += Math.pow(Math.abs(pop[i].chroms[j].phenotype), j +
2);
                                }
                                pop[i].fitness = sum1;
                                maxFitFunc = false;
                                break;
                        case "Ackley's Path":
                                for (j = 0; j < numChroms.value; j++)
                                {
                                        sum1 += pop[i].chroms[j].phenotype;
                                        sum2 += Math.cos(2*Math.PI*pop[i].chroms[j].phenotype);
                                }
                                pop[i].fitness =  -20 * Math.exp(-0.2 *
Math.sqrt(sum1/numChroms.value))-Math.exp(sum2/numChroms.value) + 20 + Math.exp(1);
                                if (isNaN(pop[i].fitness))
                                {
                                        pop[i].fitness = 25;
                                }
                                maxFitFunc = false;
                                break;
                        case "Michalewicz":
                                for (j = 0; j < numChroms.value; j++)
                                {
                                        sum1 += Math.sin(pop[i].chroms[j].phenotype) *
Math.pow(Math.sin((j + 1) * Math.pow(pop[i].chroms[j].phenotype, 2) / Math.PI), 20);
                                }
                        default:
                                        Alert.show("Error: Invalid fitness function!");
                                        i = popSize.value - 1;
                                        scGens.value = 1;
                }
                if (pop[i].fitness > fitnessPlotYAxis.maximum)
                {
                        fitnessPlotYAxis.maximum = pop[i].fitness;
                } else if (pop[i].fitness < fitnessPlotYAxis.minimum)
                {
                        fitnessPlotYAxis.minimum = pop[i].fitness;
                }
        }
}

private function sortFitness(pop:ArrayCollection):void
// Sorts the population by fitness and calculates each roulette wheel probability
{
        if (useElitism.selected)
        {
                pop.source = pop.source.concat(oldPop.source);
        }
        var s:Sort = new Sort();
        if (maxFitFunc)
        {
                s.fields = [new SortField("fitness",false,true,true)];
        } else {
                s.fields = [new SortField("fitness",false,false,true)];
        }
        pop.sort = s;
        pop.refresh();

        var sumFitness:Number = 0;
        var maxFitness:Number = Number.MIN_VALUE;
        var minFitness:Number = Number.MAX_VALUE;
        for (var i:uint = 0; i < pop.length; i++)
        {
                sumFitness += pop[i].fitness;
                if (pop[i].fitness > maxFitness)
```

```
                {
                        maxFitness = pop[i].fitness;
                }
                if (pop[i].fitness < minFitness)
                {
                        minFitness = pop[i].fitness;
                }
        }
        // Calculate probabilities for roulette wheel
        var sumProb:Number = 0;
        for (i = 0; i < pop.length; i++)
        {
                if (maxFitFunc)
                {
                        pop[i].prob = sumProb + pop[i].fitness / sumFitness;
                } else {
                        pop[i].prob = sumProb + (maxFitness - pop[i].fitness) / (pop.length *
maxFitness - sumFitness);
                }
                sumProb = pop[i].prob;
        }
        acPlots.addItem( { meanFitness:sumFitness/popSize.value, evolution:acPlots.length,
maxFitness:maxFitness, minFitness:minFitness } );

        if (useElitism.selected)
        {
                pop.source = pop.source.splice(0,popSize.value);
        }
}

private function tabChange(event:Event):void
// Changes nomenclature based on the selected algorithm
{
        switch (algSel.selectedIndex)
        {
                case 1: // BBO
                        scGensLabel.text = "Migration Limit:";
                        popSizeLabel.text = "Number of Islands:";
                        numChromsLabel.text = "SIVs per Island:";
                        chromSizeLabel.text = "Bits per SIV:";
                        bpiLabel.text = "Bits per Island:";
                        fitFuncsLabel.text = "Habitat Suitability Function:";
                        phenLabel.text = "for SIV #:";
                        updateIProb(newPop);
                        updateEProb(newPop);
                        break;
                default:
                        scGensLabel.text = "Generation Limit:";
                        popSizeLabel.text = "Population Size:";
                        numChromsLabel.text = "Chromosomes per Individual";
                        chromSizeLabel.text = "Bits per Chromosome:";
                        bpiLabel.text = "Bits per Individual:";
                        fitFuncsLabel.text = "Fitness Function:";
                        phenLabel.text = "for Chromosome #:";
        }
}

private function togglePlot(event:LegendMouseEvent):void
// Shows/hides plot and rescales fitness axis of the graph
{
        var axisMin:Number = Number.MAX_VALUE;
        var axisMax:Number = Number.MIN_VALUE;
        if (event.item.element.visible)
        {
                event.item.element.visible = false;
                event.item.marker.alpha = 0.5;
                event.item.setStyle("color", 0xaaaaaa);
        } else {
                event.item.element.visible = true;
                event.item.marker.alpha = 1.0;
                event.item.setStyle("color", 0x000000);
        }
        for (var i:uint = 0; i < fitnessPlot.series.length; i++)
        {
```

```
                if (fitnessPlot.series[i].visible)
                {
                        for (var j:uint = 0; j < fitnessPlot.series[i].dataProvider.length; j++)
                        {
                                if
(fitnessPlot.series[i].dataProvider[j][fitnessPlot.series[i].yField] > axisMax)
                                {
                                        axisMax =
fitnessPlot.series[i].dataProvider[j][fitnessPlot.series[i].yField];
                                } else if
(fitnessPlot.series[i].dataProvider[j][fitnessPlot.series[i].yField] < axisMin)
                                {
                                        axisMin =
fitnessPlot.series[i].dataProvider[j][fitnessPlot.series[i].yField];
                                }
                        }
                }
        }
        fitnessPlotYAxis.maximum = axisMax;
        fitnessPlotYAxis.minimum = axisMin;
}

private function updateIProb(pop:ArrayCollection):void
// Updates the immigration probabilities for BBO
{
        for (var i:uint = acProbs.length; i < popSize.value; i++)
        {
                acProbs.addItem( { rank:i+1 } );
        }
        for (i = 0; i < acProbs.length; i++)
        {
                switch (ipFunc.selectedLabel)
                {
                        case "Uniform Linear":
                                acProbs[i].immigration = (i + 1) / popSize.value;
                                break;
                        case "Roulette Wheel":
                                if (pop.length > i)
                                {
                                        acProbs[i].immigration = pop[i].prob;
                                } else {
                                        acProbs[i].immigration = 0;
                                }
                                break;
                        case "Exponential":
                                acProbs[i].immigration = 1 - Math.exp( -(i+1) / popSize.value);
                                break;
                        case "Sine [-pi/2,pi/2]":
                                acProbs[i].immigration = 0.5 - Math.sin(Math.PI / 2 -
i/(popSize.value - 1)*Math.PI) / 2;
                                break;
                        case "Hyperbolic Tangent [-pi,pi]":
                                acProbs[i].immigration = 0.5 - (Math.exp(2 * (Math.PI -
i/(popSize.value - 1)*2*Math.PI)) - 1) / (Math.exp(2 * (Math.PI - i/(popSize.value -
1)*2*Math.PI)) + 1) / 2;
                                break;
                        case "Step at N/2":
                                acProbs[i].immigration = int(i >= popSize.value / 2);
                                break;
                }
                if (pop.length > i)
                {
                        pop[i].immigration = acProbs[i].immigration;
                }
        }
        acProbs.refresh();
        probPlot.dataProvider = acProbs;
}

private function updateEProb(pop:ArrayCollection):void
// Updates the emigration probabilities for BBO
{
        for (var i:uint = acProbs.length; i < popSize.value; i++)
        {
```

```
                acProbs.addItem( { rank:i+1 } );
        }
        for (i = 0; i < acProbs.length; i++)
        {
                switch (epFunc.selectedLabel)
                {
                        case "Uniform Linear":
                                acProbs[i].emigration = 1 - (i + 1) / popSize.value;
                                break;
                        case "Roulette Wheel":
                                if (pop.length > i)
                                {
                                        acProbs[i].emigration = 1 - pop[i].prob;
                                } else {
                                        acProbs[i].emigration = 0;
                                }
                                break;
                        case "Exponential":
                                acProbs[i].emigration = Math.exp( -(i+1) / popSize.value);
                                break;
                        case "Sine [-pi/2,pi/2]":
                                acProbs[i].emigration = 0.5 - Math.sin( -Math.PI / 2 +
i/(popSize.value - 1)*Math.PI) / 2;
                                break;
                        case "Hyperbolic Tangent [-pi,pi]":
                                acProbs[i].emigration = 0.5 - (Math.exp(2 * (-Math.PI +
i/(popSize.value - 1)*2*Math.PI)) - 1) / (Math.exp(2 * (-Math.PI + i/(popSize.value -
1)*2*Math.PI)) + 1) / 2;
                                break;
                        case "Step at N/2":
                                acProbs[i].emigration = int(i < popSize.value / 2);
                                break;
                }
                if (pop.length > i)
                {
                        pop[i].emigration = acProbs[i].emigration;
                }
        }
        acProbs.refresh();
        probPlot.dataProvider = acProbs;
}

private function clearPlots(onlyHidden:Boolean = false):void
// Clears all the plots from the graph
{
        var tmp:Array = fitnessPlot.series.slice(0,3);
        if (onlyHidden)
        {
                for (var i:uint = 3; i < fitnessPlot.series.length; i++)
                {
                        if (fitnessPlot.series[i].visible)
                        {
                                tmp.push(fitnessPlot.series[i]);
                        }
                }
        }
        fitnessPlot.series = tmp;
}

private function updatePhenChrom():void
{
        var newPhens:ArrayCollection = new ArrayCollection();
        for (var i:uint = 0; i < numChroms.value; i++)
        {
                if (i < acPhens.length)
                {
                        newPhens.addItem(acPhens[i]);
                } else {
                        newPhens.addItem( { chrom:i+1, from:phenFrom.text, to:phenTo.text } );
                }
        }
        acPhens = newPhens;
}
```

```
private function applyPhens():void
// Assigns phenotype ranges to the selected chromosome number
{
	if (phenAll.selected)
	{
		for (var i:uint = 0; i < acPhens.length; i++)
		{
			acPhens[i] = { chrom:i+1, from:phenFrom.text, to:phenTo.text };
		}
	} else {
		cbPhenChrom.selectedItem.from = phenFrom.text;
		cbPhenChrom.selectedItem.to = phenTo.text;
	}
}

private function exportShown():void
// Opens a window with CSV text for the shown plots
{
	var exWin:TitleWindow = TitleWindow(PopUpManager.createPopUp(this, TitleWindow, false));
	exWin.title = "Fitness Data";
	exWin.alpha = 1.0;
	exWin.width = 300;
	exWin.height = 400;
	var tarea:TextArea = new TextArea();
	tarea.percentWidth = 100;
	tarea.percentHeight = 100;
	tarea.editable = false;
	for (var i:uint = 3; i < fitnessPlot.series.length; i++)
	{
		if (fitnessPlot.series[i].visible)
		{
			tarea.text += fitnessPlot.series[i].displayName + "\n";
			tarea.text += "Evolution,Fitness\n";
			for (var j:uint = 0; j < fitnessPlot.series[i].dataProvider.length; j++)
			{
				tarea.text +=
fitnessPlot.series[i].dataProvider[j].evolution.toString() + "," +
fitnessPlot.series[i].dataProvider[j][fitnessPlot.series[i].displayName.split(":
")[1].toLowerCase()+"Fitness"].toString() + "\n";
			}
			tarea.text += "\n";
		}
	}
	exWin.addChild(tarea);
	exWin.showCloseButton = true;
	exWin.addEventListener(CloseEvent.CLOSE,function():void
{PopUpManager.removePopUp(exWin);});
	PopUpManager.centerPopUp(exWin);
}

private function openInd(event:ListEvent):void
// Opens a window that allows for browsing of chromosomes for one individual
{
	var exWin:TitleWindow = TitleWindow(PopUpManager.createPopUp(this, TitleWindow, false));
	exWin.title = "Individual Genotype Breakdown";
	exWin.alpha = 1.0;
	exWin.width = 300;
	exWin.height = 200;
	var tChrom:Text = new Text();
	tChrom.text = phenLabel.text.substring(phenLabel.text.indexOf(" ")+1);
	var cbChrom:ComboBox = new ComboBox();
	cbChrom.dataProvider = acPhens;
	cbChrom.labelField = "chrom";
	cbChrom.addEventListener(Event.CHANGE,function():void { theGen.text =
oldPop[event.rowIndex].chroms[cbChrom.selectedIndex].genotype; thePhen.text =
oldPop[event.rowIndex].chroms[cbChrom.selectedIndex].phenotype; });
	var tGen:Text = new Text();
	tGen.text = "Genotype:";
	var theGen:Text = new Text();
	theGen.text = oldPop[event.rowIndex].genotype.substring(0,chromSize.value);
	var tPhen:Text = new Text();
	tPhen.text = "Phenotype:";
	var thePhen:Text = new Text();
	thePhen.text = oldPop[event.rowIndex].chroms[0].phenotype.toString();
```

```
        exWin.addChild(tChrom);
        exWin.addChild(cbChrom);
        exWin.addChild(tGen);
        exWin.addChild(theGen);
        exWin.addChild(tPhen);
        exWin.addChild(thePhen);
        exWin.showCloseButton = true;
        exWin.addEventListener(CloseEvent.CLOSE,function():void
{PopUpManager.removePopUp(exWin);});
        PopUpManager.centerPopUp(exWin);
}

private function dataTipFunc(hd:HitData):String
// Formats the dataTip of the graph
{
        var fitness:String = "";
        if (hd.item.hasOwnProperty("meanFitness"))
        {
                fitness = hd.item.meanFitness.toString();
        } else if (hd.item.hasOwnProperty("maxFitness"))
        {
                fitness = hd.item.maxFitness;
        } else if (hd.item.hasOwnProperty("minFitness"))
        {
                fitness = hd.item.minFitness;
        }
        return "<b>"+(hd.element as LineSeries).displayName+"</b>\nEvolution:
"+hd.item.evolution.toString()+"\nFitness: "+fitness;
}

private function concatChroms(item:Object, ...args):String
// Concatenates all the chromosomes for display in the grid
{
        var str:String = "";
        for (var i:uint = 0; i < item.chroms.length; i++)
        {
                str += item.chroms[i].genotype;
        }
        return str;
}
```