Evolutionary Algorithm Sandbox

A Web-Based Graphical User Interface for Evolutionary Algorithms

Brent G. Gardner
Department of Electrical and Computer Engineering
Cleveland State University
Cleveland, Ohio, USA
brent@ebrent.net

Dan Simon
Department of Electrical and Computer Engineering
Cleveland State University
Cleveland, Ohio, USA
d.j.simon@csuohio.edu

***Abstract*—The Evolutionary Algorithm (EA) Sandbox is an Adobe® Flex®-based graphical user interface (GUI) that provides a visual demonstration of evolutionary algorithm simulations. It allows the user to select EA parameters and algorithms (such as a basic genetic algorithm and biogeography-based optimization), run a simulation, and view the results after each generation. 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.**

***Keywords*—graphical user interface, evolutionary algorithm, genetic algorithm, biogeography-based optimization**

I. Introduction

A graphical user interface (GUI) for evolutionary algorithms (EAs), a broader term encompassing genetic algorithms (GAs) [1], can help users quickly learn the effects that different parameters and algorithms have on solving certain problems. There have been many GUIs written for EAs; the first ones appearing in the mid-1990's [2]-[7]. These GUIs were written in various languages and had many features in common. They mainly focused on GAs, allowing the user to adjust parameters, run a simulation and view the results graphically. There are also many toolboxes written for MATLAB® that include a GUI for EAs [8]-[10]. Though the capabilities of MATLAB are quite powerful, it is expensive and not easily accessible to a large audience. This inaccessibility was also common to the early GUIs, as they needed to be compiled locally or required a parent simulation software package to run.

The EA Sandbox was developed using the Flex® programming environment from Adobe®. Flex incorporates two main languages, MXML and ActionScript. MXML is a user interface markup language that uses the now common Extensible Markup Language (XML) syntax. ActionScript is a scripting language that can interact with the MXML-generated components. To one familiar with web development, these languages mimic the functionality of HTML and JavaScript, respectively. To run a Flex application, it must be compiled into a Flash® application using a free compiler included in the Flex Software Delvelopment Kit (SDK), and can be run in any web browser with the Adobe Flash Player plug-in installed.[1] The Flex SDK is open source under the Mozilla Public License, and is available for free from Adobe. However, the EA Sandbox uses advanced visualization components only included with Adobe Flex Builder™ Pro, a commercial development environment. This software is currently provided at no charge to students, faculty and staff of eligible educational institutions [11].

One example of a related Flex-based web application is the Genetic Algorithm Simulator [12], which demonstrates one example problem using a GA. It has limited functionality, but provides a basic illustration of how a GA works and allows the user to modify common settings. The EA Sandbox offers expanded functionality that includes multiple algorithms (currently a basic GA and biogeography-based optimization [BBO]), more problem functions (currently 11) and parameter settings, and population statistics graphing. It should be noted that these Flex-based applications are not meant to replace simulation software, but simply give an introduction to the field of EAs.

The goal of the EA Sandbox is to provide an easily accessible learning tool that inspires users (especially students) to learn more about EAs by giving them the ability to see an EA simulation in action. The EA Sandbox includes many of the common features included in other GUIs, and provides a framework for users to add their own EAs and problem functions. This paper describes the design and usage of the EA Sandbox, and also includes a tutorial for extending the application. Many of the common EA terms in this paper are explained by the references, and it is assumed that the reader is familiar with EAs in general.

II. Software Design

A. Interactive Graphical Interface

The graphic components of the interface are written in MXML and the routines executed when a user interacts with the components are written in ActionScript. Every action a user can commit to the interface (i.e. click, drag, or mouse-over) can be assigned a routine. Buttons are assigned to call an ActionScript routine when they are clicked. Another

This work was supported by Grant 0826124 in the CMMI Division of the Engineering Directorate of the National Science Foundation.

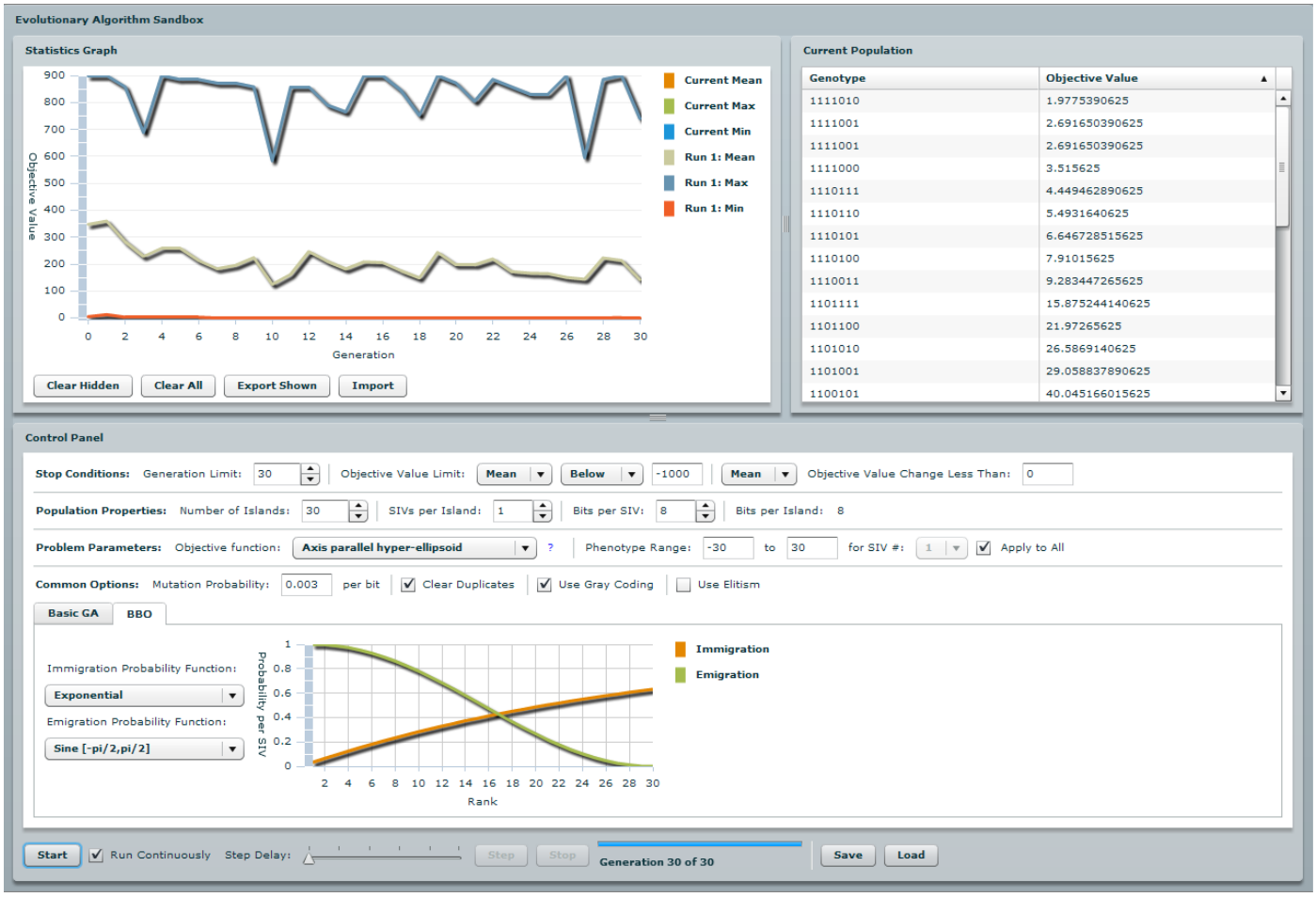

Figure 1. Screenshot of the EA Sandbox

example is when the user selects a different algorithm tab, the parameter labels are changed to reflect the nomenclature of the selected algorithm.

The user interface is divided into three sections, as seen in Fig. 1: a graph of objective value plots, a current population table, and a control panel. The graph section contains plots of the statistics of a population (mean, maximum, and minimum objective value) versus the generation index of the population. This section also provides buttons to control the display of the plots, export plots to text, and import plots from text to the graph. The current population table at the upper right of the GUI lists all of the individuals of the current generation of the population, including their genotypes and objective values. These sections may be viewed as "tiled" or "tabbed", depending on the resolution of the computer monitor. A tiled view is recommended for high resolutions and a tabbed view is recommended for low resolutions. This option is presented upon loading of the application.

The control panel at the bottom of the GUI is divided into three sections as well: evolution parameters, algorithm parameters, and simulation control. The evolution parameters include stop conditions, population properties (size of population and size of genotype, i.e., problem dimension), problem parameters (objective function and phenotype range), and other common evolutionary simulation options. The algorithm to be used in the simulation is selected from a tabbed list, with additional options for each algorithm under each tab. Currently, the available algorithms include a basic GA and biogeography-based optimization (BBO) [13]. The simulation control bar at the bottom of the control panel allows the user to start, stop, and control the speed of the simulation so that evolutionary progress can be viewed in real time if desired. There are also buttons that save and load all the simulation parameters.

B. Simulation

The EA Sandbox uses bit strings for genotypes (as opposed to quaternary DNA in [12]) which are converted to decimal phenotypes during evaluation of the objective function. The EA Sandbox could be modified to support other genotype encodings. Major modifications are further discussed in Section IV.

The EA simulation is started by the user clicking the "Start" button. This will call the ActionScript routine *startEA*, which begins the EA process as depicted in Fig. 2. First, a population is randomly initialized according to the parameters set in the control panel. This population is evaluated according to the selected objective function and

assigned the resulting value using the *evalPop* routine. The objective value can be defined as cost when the goal is to minimize, or fitness when the goal is to maximize (especially in the case of GAs). The objective functions currently implemented in the EA Sandbox are a subset of the single objective functions used in [9] which include: De Jong's function 1, axis parallel hyper-ellipsoid function, rotated hyper-ellipsoid function, moved axis parallel hyper-ellipsoid function, Rosenbrock's valley (De Jong's function 2), Rastrigin's function, Schwefel's function, Griewangk's function, sum of different powers, Ackley's Path function, and Michalewicz's function.

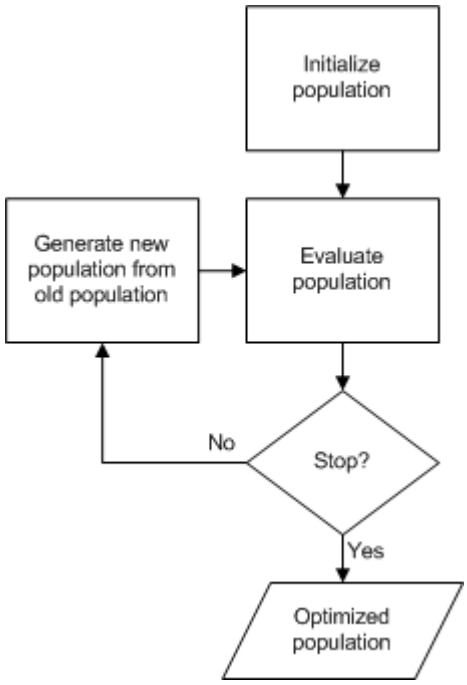

Figure 2. Process flow of an evolutionary algorithm

The next step in the EA process is to create a new population from the old population, using methods defined by the selected algorithm. The methods use the objective value of each individual to determine which individuals should be used to generate new individuals. The new population is then optionally processed using the "common options" parameters, described in Section III. This is all accomplished by the *stepEA* routine. Each time this step completes, it is called a generation. Steps are executed by the user clicking the "Step" button, or automatically if the "Run Continuously" option is checked. After each generation, the statistics plots will be updated on the graph and the current population table will display the new population. At the end of the *stepEA* routine, the *stopConditions* routine is called to check if any of the stop conditions have been met (see Section III-C), and if so, stops the simulation. The simulation will also stop if the "Stop" button is clicked by the user. After a simulation has been stopped, the *addPlots* routine is executed which "stores" the plots on the graph.

It should be noted that ActionScript executes in the user's browser, and therefore can become a strain on the user's computing resources. For this reason, certain paramters have been restricted to reasonable maximum limits: 100 generations, a population size of 100, 16 chromosomes per individual, and 16 bits per chromosome. Otherwise, the simulation could cause the user interface to "freeze" due to the intense computation required by the simulation. These limits can be easily modified by changing the *maximum* attribute for each parameter control in the *easandbox.mxm*l file. It should be noted that Adobe Flash Player running in Windows® Internet Explorer® seems to execute simulations faster than in a plug-in based browser. For complex simulations requiring large populations or generations, stand-alone simulation software should be used. Again, the EA Sandbox is meant to be a learning tool and starting point for further exploration.

III. EA Sandbox Usage

Usage of the EA Sandbox is meant to be very simple and self-explanatory. The interface provides many tool-tips which pop-up during a mouse-over of certain items, such as checkboxes and buttons. These tool-tips contain a detailed description of the item and how selecting the item will affect the interface or simulation. In this section, each interface item and simulation option will be described in detail.

A. Current Population Table

The current population table lists all the individuals, which are also called islands in the case of BBO, of the current generation. For each individual, its concatenated genotype bit string and corresponding objective value are displayed. The genotype and phenotype for each chromosome (or called suitability index variable [SIV] in the case of BBO) of an individual can be viewed by double-clicking on the listing. This will open a window which includes a drop-down box containing the index of each chromosome. When a user selects a chromosome, its genotype and phenotype are displayed. This window can be closed by clicking the "X" in the upper right hand corner.

B. Statistics Graph

The statistics graph contains plots of the mean, maximum, and minimum objective values versus generation for each simulation. The current simulation (or "run") statistics are displayed first in the legend, and completed runs are displayed below the current run. Each run is given a sequential number, which can be reset only by reloading the application in the browser. Clicking on a plot's legend item, the user can hide or show the plot on the graph. The graph will scale its y-axis after each show or hide, allowing the user to zoom in on a specific plot. Data points can also be viewed by mousing-over points on the plots.

Clicking the "Clear Hidden" button will remove hidden plots from the graph permanently. Clicking the "Clear All" button will remove all plots from the graph permanently. Clicking the "Export Shown" button will open a window with comma-separated values for each of the completed simulation plots shown. This data may be copied to a text editor and saved for import. Clicking the "Import" button will open a window into which saved text can be pasted.

Clicking the "Import" button within this window will add the saved plots to the graph.

C. *Control Panel*

As mentioned before, the control panel is divided in the three sections: evolution parameters, algorithm parameters, and simulation control. The first set of evolution parameters are stop conditions. If any of the stop conditions are met, the simulation will stop. The first and most common stop condition is the generation limit. The other stop conditions compare the population statistics to a limit or magnitude of objective value change between generations.

The next set of evolution parameters are the population properties. These define the population size, number of chromosomes per individual, and bits per chromosome. Limits are currently imposed on these settings to insure stable execution of the simulation, as mentioned in Section II.

The problem that the EA attempts to solve is defined by the problem parameters. The objective function is selected from the drop-down list, and the definition of each can be found by clicking the "?", which links to [8]. The domain of the problem is defined by the phenotype ranges, which can be applied individually to each chromosome, or applied to all chromosomes if the "Apply to All" option is checked.

The next set of parameters is common to most EAs. Mutation probability is the probability that a bit will be complemented in a genotype. Clearing duplicates will replace duplicate individuals in the population with random individuals, which are not, however, guaranteed to be unique in the population. Using Gray coding will encode the binary genotypes in Gray code, also known as reflected binary code. Using elitism will copy the best individual of the old population to the new population at each generation.

The second section of the control panel involves the selection of the algorithm and its parameters. With Basic GA selected, properties of the algorithm are displayed along with two options: "Crossover Probability (per chromosome)" and "Always Use Stud". A crossover probability of one guarantees crossover, as the single-point crossover will not occur at either end of the chromosome. Always using the stud will use the fittest individual of the population as a parent in the generation of a new individual [14]. With BBO selected, the immigration and emigration probability functions can be chosen. A graph is also displayed, plotting the probabilities versus rank. Plots representing relative cost or fitness will change each time the population changes.

The last section of the control panel allows the user to control the simulation. Clicking the "Start" button will initialize the population and evaluate each individual's objective value. The next generation will be created only if the "Run Continuously" option is checked, or if the "Step" button is clicked. The "Step Delay" slider will adjust the delay time between generations if the simulation is running continuously. Any of the settings can be changed between generations. This allows the user to try different parameters that can "hone in" on the best solution. For example, one set of parameters may find an optimal solution range, but not find the best solution. Changing the parameters during a simulation may allow the algorithm to find the best solution faster than if the parameters are not changed.

The simulation will stop if any of the stop conditions are met, or the user clicks the "Stop" button. The "Save" and "Load" buttons can be used to export and import control panel parameters, similar to importing and exporting plots.

IV. Software Extendibility

The EA Sandbox not only allows users to compare existing parameters, algorithms, and problem functions, but can also be extended to include custom parameters, optimization algorithms, and problem functions. Using the current source code as a framework, the user can customize the EA Sandbox to demonstrate a unique evolutionary algorithm simulation. This section provides some guidelines for extending the EA Sandbox.

A. *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 objective function drop-down list in the control panel. This can be accomplished by editing the *init* routine located in the *easandbox.as* file. Fig. 3 shows the location and examples of code for existing objective functions. The new function name must be different than existing functions.

```
private function init():void
{
    ...
    objFuncs.push("Rastrigin");
    objFuncs.push("Schwefel");
    objFuncs.push("Griewangk");
    objFuncs.push("Sum of different power");
    objFuncs.push("Ackley's Path");
    objFuncs.push("Michalewicz");
    ...
}
private function evalPop(pop:ArrayCollection):void
{
    for (var i:uint = 0; i < pop.length; i++)
    {
        var j:uint;
        var k:uint;
        var sum1:Number = 0;
        var sum2:Number = 0;
        switch(objFunc.selectedItem)
        {
            case "De Jong 1":
                for (j = 0; j < numChroms.value; j++)
                {
                    sum1 += Math.pow(pop[i].chroms[j].phenotype,2);
                }
                pop[i].objValue = sum1;
                maxObjFunc = 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].objValue = sum2;
                maxObjFunc = false;
                break;
                ...
}
```

Figure 3. Sample code for adding objective functions

Next, the function must be added to the *evalPop* routine, also located in the *easandbox.as* file as shown in Fig. 3. The mathematical expression for the function should be inserted as a *case* statement in the *switch* code block. The *maxObjFunc* variable is set to *true* if the function is to be maximized or set to *false* if the function is to be minimized. Additional intermediate variables like *j*, *k*, *sum1*, and *sum2* must be defined before the *switch* statement.

B. Adding an Algorithm

Adding a new optimization algorithm to the EA Sandbox involves three steps. The first is to add the algorithm to the *TabNavigator* control in the *easandbox.mxml* file as a *VBox* or *HBox* element, with the name of the new algorithm as the *label* attribute. Using the "Basic GA" and "BBO" algorithms as a guide, additional controls can be added to the new algorithm as in Fig. 4. These controls will allow the user to adjust algorithm parameters, such as the crossover probability in the case of the basic GA. The *id* attribute of each MXML control is used to retrieve its value in ActionScript for use in the simulation. The new algorithm will be displayed in the order placed in the *TabNavigator* code block. The *tabChange* routine located in the *easandbox.as* file will execute each time a tab is changed. This routine can be used to change the nomenclature of options in the control panel, if desired.

```
<mx:TabNavigator id="algSel" change="tabChange(event);">
    <mx:VBox label="Basic GA">
        <mx:CheckBox id="pselStud" label="Always Use Stud" />
        <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:HBox>
    </mx:VBox>
    ...
</mx:TabNavigator>
```

Figure 4. Sample code for adding an algorithm to the GUI

Next, the algorithm needs to be added to the *stepEA* function in the *easandbox.as* file as a *case* statement in the *switch* code block of Fig. 5. The *case* value must be the same as the *label* attribute of the element added to the *TabNavigator* code block. The new algorithm may use existing routines such as: *mutatePop* (mutates the population according to the mutation probability), or *clearDups* (replaces duplicate individuals with new, random individuals). The new algorithm may also use custom routines developed by the user. After the algorithm has generated a new population (*newPop*) from the population of the previous generation (*oldPop*), the routine *evalPop,* and optionally *sortPop,* should be called to assign objective values to the new individuals and calculate the statistics of the new population.

```
private function stepEA():void
{
    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 bPop:ArrayCollection = crossOver(selectParents(oldPop));
                    for (j = 0; j < bPop.length; j++)
                    {
                        newPop.addItem(bPop[j]);
                    }
                }
                mutatePop(newPop);
                clearDups(newPop);
                evalPop(newPop);
                sortPop(newPop);
                break;
                ...
}
```

Figure 5. Sample code for adding an algorithm in ActionScript

C. Future Extensions

The nature of the EA Sandbox allows for numerous ways to improve and enhance it. Many other EAs, parameters, and options could be added. The ability for the EA Sandbox to handle symbolic problems, such as the problem used in [12], would require new controls and routines. These will be considered as development of the EA Sandbox continues.

A future use of the EA Sandbox could be to control and provide feedback from a simulation not running on the user's computer. Flex provides many means of communicating with external servers, and the EA Sandbox could easily be modified to provide this capability. Parameters could be set by a user using the EA Sandbox, sent to a server via a network connection (such as the internet), and the server would execute the prescribed simulation. Feedback, such as population statistics, could be sent from the server back to the EA Sandbox so the user can monitor the progress of the simulation. This would allow the EA Sandbox to maintain its functionality, but pass the intense computation on to a better suited tool.

V. Conclusion

The EA Sandbox was created as a visual learning tool for evolutionary algorithms. A basic set of parameters, problem functions, and algorithms were included. Simulations using these different options can be executed, the results viewed in real time, and compared to one another. Users can quickly test different options to explore their effects, as well as add their own options. The EA Sandbox application, source code, and documentation can be downloaded from http://embeddedlab.csuohio.edu/BBO.

References

[1] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Boston: Addison-Wesley, 1989.

[2] C. Jacob and A. Burghof, "NeXTGene: a graphical user-interface for GENESIS under NeXTStep," Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms, pp. 602-606, 14-16 April 1993.

[3] T. Dabs and J. Schoof, "A graphical user interface for genetic algorithms," Report no. 98, Universität Würzburg, Würzburg, Germany, 1995.

[4] Y. Yoshida, N. Adachi, and K. Tajima, "GAVIEW – a visualisation tool for supporting GA simulations and analyses," Proceedings of IEEE International Conference on Evolutionary Computation, 1996, pp. 437-442, 20-22 May 1996.

[5] E. K. Burke and D. B. Varley, "A genetic algorithms tutorial tool for numerical function optimisation," Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education, pp. 27-30, 1997.

[6] A. S. Chang and F. Wu, "An extensible genetic algorithm framework for problem solving in a common environment," IEEE Transactions on Power Systems, vol. 15, no. 1, pp. 269-275, Feb 2000.

[7] Y. Liao and C. Sun, "An educational genetic algorithms learning tool," IEEE Transactions on Education, vol. 44, no. 2, p. 210, May 2001.

[8] A. Li and K. P. Wong, "Animating the evolution process of genetic algorithms," Lecture Notes in Computer Science, vol. 1585, pp. 341-348, 1998.

[9] H. Pohlheim, "GEATbx – Genetic and Evolutionary Algorithms Toolbox in Matlab," 2006. [Online]. Available: http://www.geatbx.com/.

[10] The MathWorks, Inc., "Optimization Toolbox – MATLAB," 2009. [Online]. Available: http://www.mathworks.com/products/optimization.

[11] Adobe Systems Inc., "Adobe Flex Builder 3 Pro for Education," 2008. [Online]. Available: http://freeriatools.adobe.com/flex.

[12] T. Fendall, "Genetic Algorithm Simulator in Flex," 2007. [Online]. Available: http://www.munkiihouse.com/?p=7.

[13] D. Simon, "Biogeography-based optimization," IEEE Transactions on Evolutionary Computation, vol. 12, no. 6, pp. 702-713, December 2008.

[14] W. Khatib and P. Fleming, "The stud GA: A mini revolution?," in Parallel Problem Solving from Nature (A. Eiben, T. Back, M. Schoenauer, and H. Schwefel, eds.), Springer, 1998, pp. 683-691.